Context-Aware and Secure Workflow Systems

Ph.D. Thesis

Hind Mobtel Alotaibi

Software Technology Research Laboratory

De Montfort University

U.K.

July 2012

ABSTRACT

Businesses do evolve. Their evolution necessitates the re-engineering of their existing "business processes", with the objectives of reducing costs, delivering services on time, and enhancing their profitability in a competitive market. This is generally true and particularly in domains such as manufacturing, pharmaceuticals and education).

The central objective of workflow technologies is to separate business policies (which normally are encoded in business logics) from the underlying business applications. Such a separation is desirable as it improves the evolution of business processes and, more often than not, facilitates the re-engineering at the organisation level without the need to detail knowledge or analyses of the application themselves. Workflow systems are currently used by many organisations with a wide range of interests and specialisations in many domains. These include, but not limited to, *office automation, finance and banking sector, health-care, art, tele-communications, manufacturing* and *education*.

We take the view that a workflow is a set of *"activities"*, each performs a piece of functionality within a given *"context"* and may be constrained by some security requirements. These activities are coordinated to collectively achieve a required business objective. The specification of such coordination is presented as a set of "execution constraints" which include *parallelisation* (concurrency/distribution), *serialisation, restriction, alternation, compensation* [1] and so on.

Activities within workflows could be carried out by humans, various software-based application programs, or processing entities according to the organisational rules, such as meeting deadlines or performance improvement. Workflow execution can involve a large number of different participants, services and devices which may cross the boundaries of various organisations and accessing variety of data. This raises the importance of

- *context* variations and *context-awareness* and
- security (e.g. access control and privacy).

The specification of precise rules, which prevent unauthorised participants from executing sensitive tasks and also to prevent tasks from accessing unauthorised services or (commercially) sensitive information, are crucially important. For example, medical scenarios will require that

[1] Compensation is a mechanism by which previously completed activities can be undone or compensated (following the logic defined by the application) when a subsequent failure occurs.

- *only authorised doctors are permitted to perform certain tasks,*
- *a patient medical records are not allowed to be accessed by anyone without the patient consent and*
- *that only specific machines are used to perform given tasks at a given time.*

If a workflow execution cannot guarantee these requirements, then the flow will be rejected. Furthermore, features/characteristics of security requirement are both temporal- and/or event-related. However, most of the existing models are of a static nature – for example, it is hard, if not impossible, to express security requirements which are

- time-dependent (e.g. *A customer is allowed to be overdrawn by 100 pounds only up-to the first week of every month.*
- event-dependent (e.g. *A bank account can only be manipulated by its owner unless there is a change in the law or after six months of his/her death*)[2].

Currently, there is no commonly accepted model for secure and context-aware workflows or even a common agreement on which features a workflow security model should support. We have developed a novel approach to design, analyse and validate workflows. The approach has the following components:

[2]Here, *change in the law* and *death* are two events, and the later one has a timing-dependency. Once one of these events occurs, the current policies must be suspended and new ones have to be applied/enforced.

- A modelling/design language (known as $\mathcal{CS}-\mathcal{F}low$). The language has the following features:
  - support concurrency;
  - context and context awareness are first-class citizens;
  - supports mobility as activities can move from one context to another;
  - has the ability to express timing constrains: delay, deadlines, priority and schedulability;
  - allows the expressibility of security policies (e.g. *access control* and *privacy*) without the need for extra linguistic complexities; and
  - enjoy sound formal semantics that allows us to animate designs and compare various designs.
- An approach known as *communication-closed* layer is developed, that allows us to serialise a highly distributed workflow to produce a semantically equivalent *quasi-sequential* flow which is easier to understand and analyse. Such re-structuring, gives us a mechanism to design fault-tolerant workflows as layers are atomic activities and various existing forward and backward error recovery techniques can be deployed.
- Provide a reduction semantics to $\mathcal{CS}-\mathcal{F}low$ that allows us to build a tool support to animate a specifications and designs. This has been evaluated

on a Health care scenario, namely the Context Aware Ward ($\mathcal{CAW}$) system. Health care provides huge amounts of business workflows, which will benefit from workflow adaptation and support through pervasive computing systems. The evaluation takes two complementary strands:

- provide $\mathcal{CS}-\mathcal{F}low$'s models and specifications and
- formal verification of time-critical component of a workflow.

Contents

List of Figures

List of Tables

DEDICATION

This thesis is dedicated to my Father **Mobtil** , and my Mother **Moudi**. It is also dedicated to all those who believe in me, most of all my family – my husband **Khalid**, my darling children: **Sara, Fai, Shaden** and **Mohammad**, without whom nothing would have ever happened .

DECLERATION

The thesis reports original work that has been undertaking by myself. It is submitted for the degree of Doctor of Philosophy at De Montfort University. The work was undertaken between October 2009 and July 2012.

ACKNOWLEDGMENTS

This research project would not have been possible without the support of many people. I wish to express my gratitude to my supervisor Dr. Amelia Platt who was abundantly helpful and offered invaluable assistance, support and guidance. And to all the members of the STRL, without whose knowledge and assistance this study would not have been successful. Dr. Ward's careful reading of manuscript has greatly strengthened its contents. For him, I'm particularly grateful.

Publications

During this work I have published initial results on the topic in three papers: [10], [11] and [8]. In these publication we only considered security issues which has been extended, and modified, in this thesis to include context and timing dimensions. A fourth paper is a journal one which will be submitted in The International Journal in Information and Software Technology [9]

Chapter 1

INTRODUCTION

Objectives

- Motivate the need for secure and context aware workflow systems and state, in a precise terms the research question.
- outline our research methodology
- Articulate our research findings
- Outline the structure of the thesis

1.1 Introduction

The separation of business logic/processes and rules and their underlying technologies that realise them, has been the concern of both industrialists and academics for many decades. Businesses evolve and their underlying technologies evolve too. Their evolution occur at rather different rates. Business evolves as a response, for example, of political pressure and the continual presence of competitive element. The response to such interplay, the evolution of business processes/logic and their supporting technologies, has to be rapid taking into account the whole development life-cycle.

The central objective and goal of workflow technology is to separate business policies (which normally have been encoded in business logic) from the underlying business applications. Such a separation is desirable as it improves the evolution of business processes and, more often than not, facilitates the re-engineering at the organization level without the need to detail knowledge or analyses of the application themselves. Workflow systems are currently used by many organisations with a wide range of interests, specialisations and application domains. These include, but not limited to, *office automation, finance and banking sector, health-care, art, tele-communications, manufacturing* and *education*. We take the view that a workflow is a set of *"activities"*, each performs a piece of functionality within a given *"context"* and may be constrained by some security requirements. These

activities are coordinated to achieve a required business objective. The specification of such coordination is presented as a set of “execution constraints" which include parallelisation (concurrency/distribution), serialisation, restriction, alternation, compensation and so on. For example, a workflow scenario for the management of a warehouse organises and controls the movement of goods around the warehouse and where about, and the way goods are being stored as efficiently and as safely as possible. These activities, and many others, are to be achieved through the precise definition, processing and realisation of many complex transactions, including goods' *shipping, receiving, putting away, picking up* and *delivering*. Activities within workflows could be carried out by humans, various software-based application programs, or processing entities according to the underlying organisational rules, such as meeting deadlines or performance improvement. Workflow execution can involve a large number of different participants, services and devices which may cross the boundaries of various organisations. This raises important issues that are related to *context-awareness* and *security*. It is important to be able to specify exact rules to prevent unauthorised participants from executing sensitive tasks and also to prevent tasks from accessing unauthorised services. For example, medical scenarios will require that only authorised doctors are permitted to perform certain tasks and that only specific machines are used in those tasks. If a workflow execution cannot guarantee these requirements, then the flow will be rejected.

Furthermore, workflows can hold and manipulate various data with different security requirements and it is important to *enforce* these requirements while the data is accessed in a workflow instance. Delegations, constraints over authorisations, audit and integrity provide additional security features. We adopt a policy-based approach in which rules are specified compositionally as policies which may be analysed and verified at run-time. Currently, there is no commonly accepted model for secure workflows or even a consensus on which features a workflow security model should support. For example, [6, 71] give typical policy-based models in which policies are “static". By static we mean that policies have *no* dependency on time or the sudden occurrence of events. Such dependencies are important as they permit policies to change at run time. It is often desirable, and sometimes crucial in many workflows, that after the elapse of a period of time or the occurrence of a particular event, new policies will be adopted and enforced rendering the old ones obsolete. Temporal/timing aspects of access control requirements are especially important in domains ranging from E-business to even military domain where the value of tactical information are highly dependent on *time*, (for example, time to start a mission and its duration) and *events* (e.g. civilian accidents, or change in troops formation). The work reported in [21, 23] has recognised the need for temporal/timing-dependent policies. However, these models and others lack compositionality and efficient mechanisms for enforcing them at run-time. Furthermore, activities within a workflow may be performed within

various *contexts*. These contexts could be different platforms (hardware, operating systems, communication speed, etc.) and/or at different locations. Sensing the particular context information, may alter the security constraints and hence an appropriate execution platform may be chosen. In summary, Current workflow design languages and their supporting management systems do not handle these two aspects, namely security- and context-aware.

1.2 Research Question

How can we design, develop and build a context-aware, secure workflow system in an integrated fashion?

the quest to answer this question has led to various sub-questions which needed to be addressed. These are

1. What is the “nature" of *context* in workflow systems?

2. What are the various considerations or dimensions of “security" in the presence of mobility and context-awareness?

3. How do we “model" these system, with attributes such as being highly distributed, context-aware, time-dependent, allow mobility and being security-critical?

4. Policy-based approaches have been shown to be valuable tools, how can these policies be represented within context-awareness?

5. Being highly distributed, is there a more efficient mechanism for the design and development of context-aware, secure workflow systems that is amenable to analysis?

6. Is there a provision for animation?

This thesis will deal with these sub-questions and provides a unique mechanism for answering the main research question.

1.3 Original Contribution

We have developed a novel approach to design, analyse and continually validate workflows within any organisation. The approach has the following components:

- A design language (known as $\mathcal{CS}-\mathcal{F}low$). The language enjoys the following features:
  - support concurrency;
  - context and context-awareness are first-class citizen;
  - supports mobility as activities can move from one context to another;

  - has the ability to express timing constrains: delay, deadlines, priority and schedulability;
  - allows the expressibility of (*access control*) security policies without the need for an extra linguistic complexities; and
  - enjoys sound formal semantics that allows us to animate design and compare various designs.

- An approach known as *communication-close* layer is developed, that allows us to serialise a highly distributed workflow to produce a semantically equivalent *quasi-sequential* flow which is easier to understand and analyse. Such re-structuring, gives us a mechanism to design fault-tolerant workflows as layers are atomic activities and various existing forward and backward error recovery techniques can be deployed.

- Provide a reduction semantics to $\mathcal{CS}-\mathcal{F}low$ that allows us to build a tool support to animate a specifications and designs. This has been evaluated on a Health care scenario, namely the $\mathcal{CAW}$ system.

1.4 Research Methodology

Our methodology follow a typical software engineering steps. These are

- Engaging critically with the existing literature and tools support for work-

flow systems and various applications. This has led to a clear articulation of the research question and scope our efforts to efficiently solve the rising issues **(Chapter 2)**.

- To gain further understandings of the way we can solve these issues, we model the situation through a computational model which is then formalised and supported by a notation for design and analysis. This has resulted in $\mathcal{CS}-\mathcal{F}low$ (**Chapters 3-6**).

- To demonstrate that our techniques are valid we have illustrated it in a number of small case study and end with a realistic case-study from the healthcare domain (**Chapter 7**).

1.5 Measure of Success

Measuring success has two components:

1. Demonstrate that our research questions have been resolved and answered: Chapter 3-6;

2. show that our approach performs better than other existing ones: Chapter 7.

1.6 Thesis Outline

The thesis is organised in some eight chapters each has its objectives and rationale[1]. The rationale of all Chapters are thus given below. Its structure is depicted in Figure(1.1) below In Chapter 2, we provide

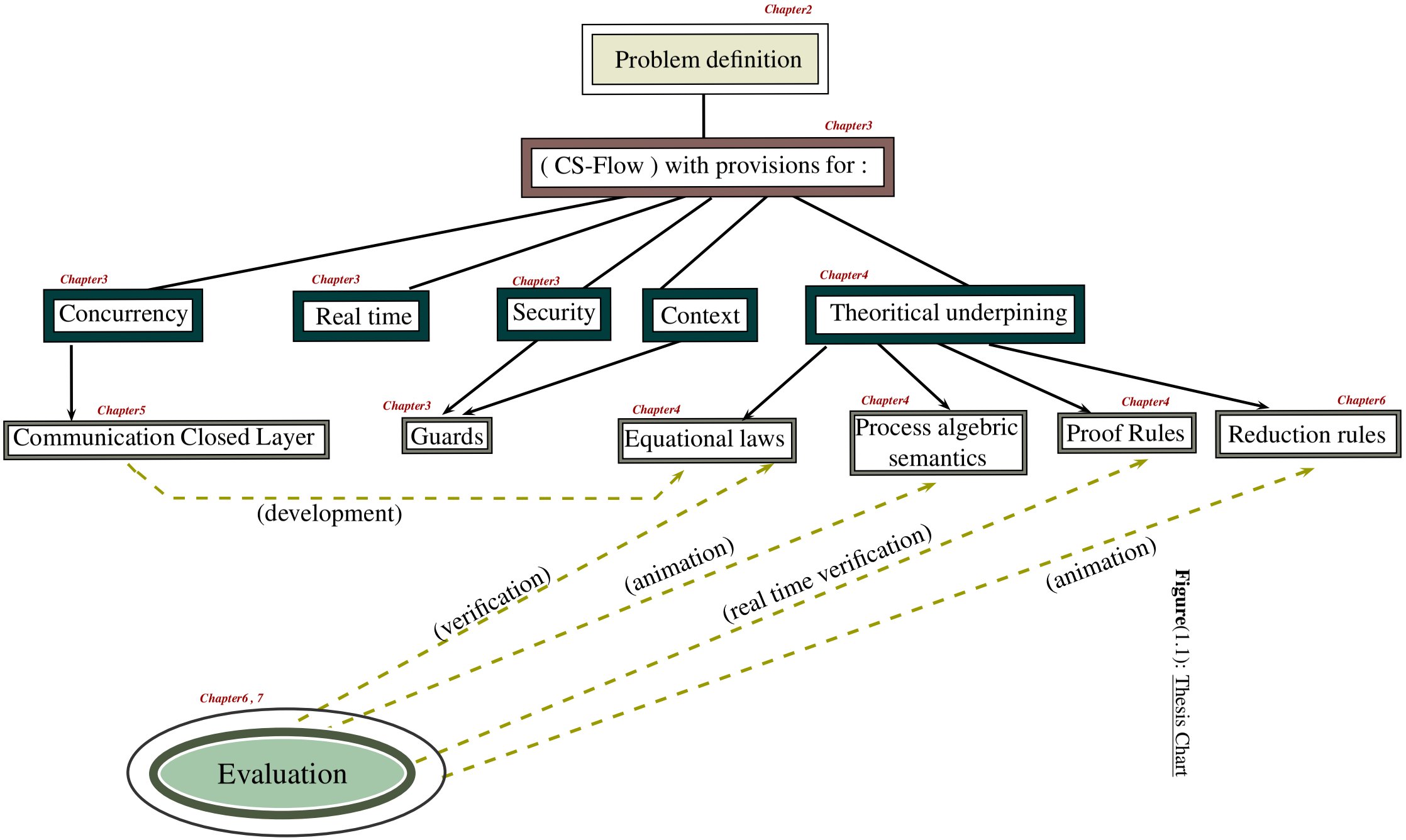
- critical engagement with previous work on Workflow principles and existing systems and
- motivate research questions and define context

In Chapters 3, 4 and 5 give details of a computation model for our Secure, Context aware Workflow systems. In particular, we

- provide an informal description of the computational model for a context-aware and secure workflow systems,
- describe our design language, $\mathcal{CS}-\mathcal{F}low$, and how it could be used,
- give a denotational (specification-oriented semantics in a process algebraic style) for $\mathcal{CS}-\mathcal{F}low$,
- give an algebraic characterisation of $\mathcal{CS}-\mathcal{F}low$, and
- provide the concept of closed layers and give a development process that supports it.

[1] A reader who is familiar with issues in Workflows and security can skip Chapter 2.

Chapter 6 outlines the design of a tool that serves as a proof-of-concept towards the executability of $\mathcal{CS}-\mathcal{F}low$. Chapter 7, provides a realistic case study from health care domain as a means of evaluating our approach. Chapter 8, summaries our findings and concludes with some future work.



Figure(1.1): Thesis Chart

Chapter 2

WORKFLOW SYSTEMS

State-of-the-Art

Objectives

- Critical engagement with previous work on workflow principles and existing systems
- Motivate research questions and define context

2.1 Introduction

A business process is a set of activities that are (appropriately) arranged to achieve a given business goal. These arrangements are commonly structured and many of these processes are traditionally well understood, predictable, repeatable and have real-time constraints. The activities in these processes are normally distinct, and the control between them flows in a well defined manner and decisions involved are simple, clear and made in a deterministic fashion. However, with the advent of ubiquitous computing environment, these traditional tasks have the extra requirement that they must decide on the next service according to the user's situation (known as *contextual* information) which is continually and dynamically changing. Most of the popular modelling techniques have no consideration/support for dealing with this contextual information (e.g. [127], [70], and [26]). In this thesis we take the view that workflow systems are inherently **context-aware** and the fact that it is highly distributed and their activities cross the boundaries of many organisations lend themselves to be **security-critical**. This is in addition to the fact that some of its activities must satisfy variety of **timing constraints**. In this Chapter we give a general overview of workflows and their importance in everyday businesses and enterprises. Simply stated, a workflow has

- *a goal*,
- *an input*,

- *an output*,
- *a number of activities which are performed in some order*,
- *may have some timing constraints*,
- *uses variety of resources of different specifications*,
- *may affect more than one organisation unit* and
- *creates some value for users*[1].

Modern workflow systems cross the boundaries of organisations, each has its own security requirements, policies and constraints. Even within one organisation, activities in a workflow systems may be executed, in one of its instances, within a platform but in another instance it may be executed or performed on a different platform with completely different environment. Indeed it may not even be automated. In this chapter we review current techniques and technologies and demonstrate the lack of these consideration.

The review concentrates on three aspects: Workflow Management Systems and their de facto standard (Sections (2.2) and (2.3)), Security requirements (Section(2.4)) and Context-Aware system and Adaptation (Section(2.6)).

[1] "value" here refers to the benefit of the "output". This will depends on the user(s) of the workflow (which may include other workflows). "value" may be *customer satisfaction, gaining market lead, etc.* "value" may be a transitional value that may be needed to achieve the overall goal of the workflow. For example `skip` is an activity/workflow with a transitional value.

2.2 Workflow and Workflow Management Systems

Workflow is often related to Business Process Re-engineering (BPR). BPR involves the following steps:

- *assess the need for change* – either to improve efficiency or response to economic/political pressure, etc.
- *analyse existing processes and their dependencies* – to establish if, for example, they still meet business goals.
- *construct new model(s) of current and future situation need (including environment)*, and
- finally *realisation* of all fundamental business processes, and/or any other (auxiliary) business entities.

A workflow system is defined as

> *The computerised facilitation or automation of a business process, in whole or part.* [65]

There is a direct correlation between workflows and BPR. Sometimes, workflow implementations form a part of a BPR exercise, yet not all BPR operations may result in the implementations of workflows. However, workflow technologies provides adequate solution as there is a clear separation between business logic and

its supporting IT infrastructure. This separation is rather important as it allows rapid changes to the rules that define the business process. In order to manage the sequence of work activities and determine the invocation of the appropriate resources (human or IT resources) needed for the activities, a **Workflow Management System** is needed. It is a fast evolving technology which is being exploited by businesses in a variety of industries and organisations, including the office environment where the number of operations that are conducted by employees are large such as insurance, banking, legal and general administration, etc. It is also applicable to some classes of industrial, manufacturing, health service and military applications. The Workflow Reference model defines Workflow Management System as:

> *A system that completely defines, manages and executes "workflows" through the execution of software whose order of execution is driven by a computer representation of the workflow logic.* [65]

Due to its continual popularity and driven by the need to provide support for BPR and the need for requirements *integration* that is resulting from the wide-range of specialisation of products and the variety of markets and domains, the Workflow Management Coalition (WfMC) and the Object Management Group (OMG) have provided a common framework within which interoperability and communication standards are developed that allow the coexistence of the various workflow

products.

2.3 Workflow Reference Model

There are some common characteristics which workflow management products share. These characteristics are believed to potentially enable systems to achieve a level of *interoperability* through the use of common standards of various functions. As a result, the WfM Coalition (WfMC) has been formed to identify these functional areas and develop appropriate specifications to be implemented within any workflow products. Their ultimate goal is to improve the effective development of workflow technologies across the IT market to the benefit of their vendors and users.

The WfMC Reference Model takes a rather general view of workflow management, and outlines a common model that intends to accommodate the various implementation techniques and their underlying technological infrastructure and operational environments. The standard was developed by considering various structures of generic workflow applications. These structures contain a number of generic components which communicate with each other in a number of defined ways; different technologies may exhibit different capabilities within each of these generic components. The process of achieving interoperability is com-

monly done by establishing a set of standardised interfaces as well as precisely defining data exchange formats between such components. This is followed by the construction of a number of distinct interoperability scenarios by reference to these interfaces, identifying various levels of functional conformance as appropriate to the range of existing technologies. There are three functional areas, at the *highest level*, that all WfM systems provide support for, [65], see Figure 2.1:

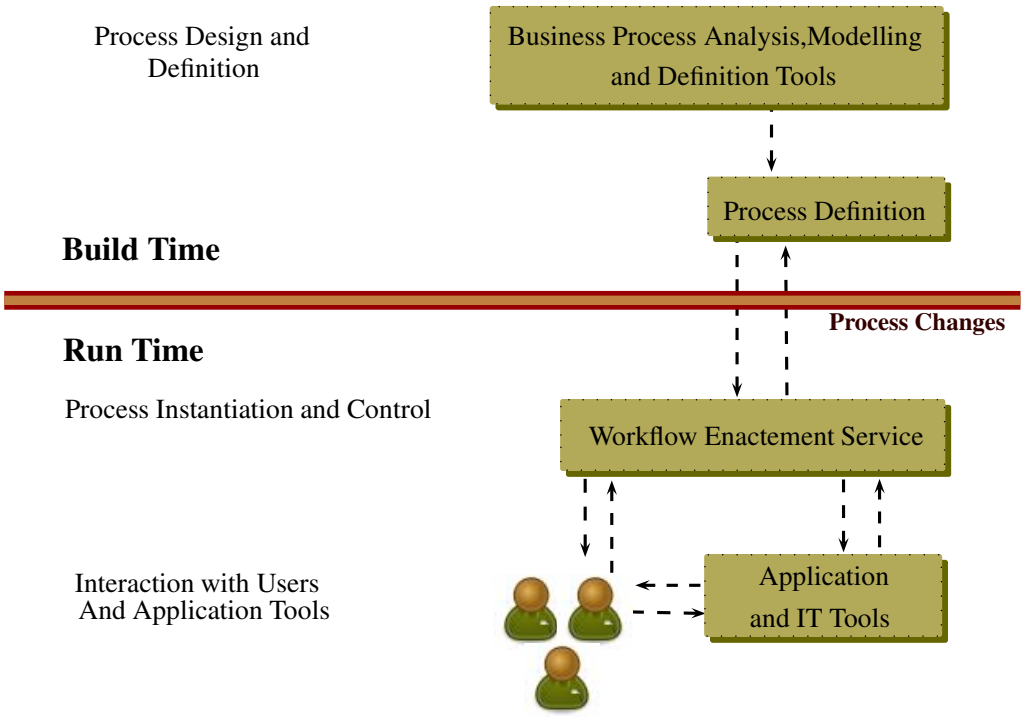


Figure 2.1 – Workflow Process [65]

- **Build-time** functions. This is concerned with defining, and possibly modelling, the workflow process and its activities. This support is provided at

the design stage of a workflow. It is the phase in which the goal is translated into technical design of the workflow. I.e. a business process is translated from an informal requirement document into a formal, design definition. This is done using one or more of the existing analysis, modelling and system techniques (e.g. [26], [127], [70]). The resulting specification/description is known as either a *process model, a process template, process metadata*, or *a process definition*, e.g. [26, 65]. This is the phase that we are concerned with in this thesis and which both **security**, **context** and **timing** requirements are specified, designed and analysed. The result of this process is a description of a number of *discrete* activity steps, with associated computer and/or human operations and rules governing the execution of the process through the various activity steps. The process description may be expressed in textual or graphical form or in a formal language notation (similar to $\mathcal{CS}-\mathcal{F}low$ (cf. next Chapter). Dynamic alterations to process description from the run-time operational environment can be specified.

- **Run-time control** functions. This phase is responsible of the actual execution of the process description that is resulted from the above phase. It is concerned with the *management* of the execution of the workflow processes in an operational setting and the orchestration (e.g. *sequencing*) of the various activities to be handled as part of each process. This may involve

various static/dynamic scheduling algorithms. Major challenge which we address in the thesis is that modern workflows is characterised by *mobility* and *context* which place extra constraints at design level.

- **Run-time interactions**. Interactions here are with human users as well as with any IT application tools for processing the variety of activity steps in any instance of a workflow. One of the challenges in this facility is that at each instance, the underlying infrastructure and the IT support (e.g. platforms) can (and many cases do) change.

  These run-time process control functions act as the linkage between the process as modelled within the process specification (the design phase) and the process as it is seen in the real world (i.e. its implementation), reflected in the runtime interactions of users and IT application tools.

The core component towards realising these functionalities is the realm of workflow *management control* software (or *engine*),which is often widely distributed and are executed on a variety of platforms. In summary, it is responsible for the

- creation and deletion of processes (and in our case in the present thesis the ability to change context and security policies);
- controlling of the activity scheduling within an operational process and varying contexts/environments; and

- interaction with various application tools or human resources.

In addition to specifications for process definition, data and its exchange protocols, WfMC, [65], provides a wide range of interfaces. These include interfaces to support:

- interoperability between different workflows;
- communication with a wide variety of IT application types;
- interaction with user through interface functions; and
- system monitoring and provides metric functions to facilitate the management of composed/integrated workflow application environments

2.3.1 Critique of standards

We have outlined the attempt that WfMC made to standardised the way workflow management systems should be designed and developed. The standard follows a typical software engineering life cycle and identified the typical issue such life cycle has. However, the standard has missed the opportunities to identify crucial issue relating to security and contexts. There is no provision for either and current/modern workflow system faces these issues. This is the topic of the subsequent chapters in this thesis.

2.4 Security Requirements

2.4.1 General security Requirements

As we mentioned early that complex day to day workflows in large enterprises and organisations are managed and facilitated by WfMSs. Fundamental and major issues with current WfMSs are that they often use heterogeneous and widely distributed hardware and software systems to execute a set of activities within a given workflow. This gives rise to decentralised security requirements [2] and the need to establish variety of mechanisms that ensure the correct enforcement of these requirements. Indeed this in turn need to be carefully and adequately *managed*. Since security is an essential and integral part of workflows, the WfMS has to manage the execution of workflows in a secure manner. In particular, a robust secure workflow model is needed to (see for example, [79], [80] and [141])

- allow controlled access of data objects and ensure their privacy and integrity,
- secure execution of tasks in the presence of variety of contexts, and
- efficient management and administration of security requirements.

For critical and strategic applications, major concerns regarding the threats against the security properties/requirements, in particular, *integrity* and *confidentiality* of

[2] Our treatment to *Requirements* encompasses the notion of policy. In fact, at a formal level, a requirement and a policy are indistinguishable.

data and processes. These properties include:

- **Integrity** which prevents the unauthorised modification of information [68]. *Integrity* here ensures the correctness and appropriateness of the content of a piece of information in a given context – a piece of information may be appropriate for a given user in one context but not so in another. Integrity is also related to the *legitimate* pattern of operations in data accesses (hence policies that govern integrity requirements are intrinsically *history-preserving*, [74,75]). In a workflow, no data should be modified or corrupted by unauthorized activities.

- **Authorisation** is the process of identifying to the system the various functions which a user may undertake [68]. In a role-based authorisation, particular privileges may be associated with certain roles to execute activities and/or access certain resources. In a workflow, authorisation means that no data or resource is accessed by unauthorised activities at anytime. Once again, the roles of a user may be dependent on the context in which the user finds him/herself in.

- **Availability** prevents the unauthorised *withholding* of information or resources [68]. The resource should be available at hand within a specific time frame (session) once it is needed. There is a clear interplay between availability and reliability of information. Here we are only concerned with

availability in the sense that, in a workflow, data or resource(s) must be at the activity's disposal when needed for *legitimate* use.

In order to ensure these three security properties of a workflow, a WfMS must manage and protect all the information of a workflow from *build-time* (or at a design phase) to run-time within a secure framework.

As a security framework has to prevent any unauthorised modification of data and to enforce the legitimate pattern of operations in data accesses by an activity, the WfMS hence needs to know *when* and *what* data and privilege(s) to be granted to this activity and at which time instant/period and also to be revoked from the activity. Further, the WfMS needs to know the conflict of interest [68, 111] for every activity (especially when roles are considered).

2.4.2 WfMC and Security Requirements

As we mentioned earlier that there is a no commonly accepted secure workflow model. We take the view that a workflow security model should support some basic requirements. These include:

- Activities are only executed by authorised users and that authorised activities are to access particular services. Provision for conflict resolution should be provided.

- Context-awareness is important as context and the surrounding environment influence the security decisions and may force mobility and adaptation.
- Due to the highly distributed nature of current workflows, a provision for the secure distributed workflow execution should be given and that the ability to specify constraints over workflow migrations and the distribution of activities.
- Adaptability is important and a security framework should allow for the modification of security policies through, for example, dynamic policy changes.

Although it is beyond the scope of this thesis, the following addition features are required for any workflow security model:

- Ability to specify privacy constrain over data which are to be manipulated by activities.
- Ensure trusted platforms over which users (human and/or activity) can be authenticated.

Workflow systems process highly valued information to those who own it and therefore it is important to protect this information from any threats whether coming from outside or inside. In a policy-based approach, a security policy typically reflects the security requirements for the workflow. A policy is a set of rules and procedures controlling the use of information, from processing, storage, to their

distribution and presentation [68]. Security requirements can describe the types and levels of protection needed for equipment, data, information, applications, and facilities to meet a security policy [68]. For example, a policy can be as simple as *a nurse at a grade X is authorised to measure blood pressure*.

Referring to the WfMC [65], we define a **secure workflow** as a computer supported business process which is able to protect it against security threats and further satisfies the security requirements as specified by the business goals. Further, a **secure Workflow Management System** (WfMS) is a workflow management system that allows the specification, management and the execution of secure workflows.

The WfMC has recommended what is known as a *security profile* in which the security services, to be applied for interoperability between two parties, are identified. The issue with the profile is in its static nature, in the sense that it is constructed at a design/build-time and does not take into account the various contexts that complex modern workflows are facing. In addition, whilst WfMC presents protocols for the interoperability between two components, the flow of authorisations amongst these components, tasks and resources during the workflow execution are not considered within the protocols.

2.4.3 Existing Models

Here we review some of the existing models that have adopted and proposed within the discipline of *Computer Security* in general.

Access Control. Role-Based Access Control (RBAC) models [122] lie at the heart of any authentication process which allocate users into roles. Some work on security in BPEL processes, e.g. [24] and [82], rely on authenticating a user through *credential* checking, but they do not specify what credentials or how they are checked.

WS-Security [2], [112], [113], [114], [110] and SAML [3] are two popular web service standards. These can be used to provide secure authentications between different services. A security token, for example in the case of WS-Security, is appended (such as certificates) to SOAP Messages. This can be viewed as a protocol to securely exchange messages between web services by providing confidentiality and integrity of SOAP messages. The SAML standard attempts to solve the Single-Sign-On problem, where a user is authenticated once by a service which is in turn gives an assertion (or a capability) that other services use to authenticate the user – the user does not need to supply his/her credentials again.

Discretionary Access Control (DAC) [117], is another access control mechanism which has been used widely in file systems. DAC was designed to model security of objects (e.g. *files* on the basis of a single subject's defined privileges (i.e. users).

Within this framework, predicates are used to describe the types of access that a subject has over an object. These predicates which has to be evaluated to "true" for granting access or "false" otherwise. Within a file system, for example, privileges can be *read, write, delete, create* and *copy*[3].

The Workflow Authorisation Model (WAM) [14, 15] is another conceptual and logical model. Execution environment is provided in which the *enforcement* of authorisation rules in activity dependency and transaction processing are emphasised. This is performed largely by using Petri Nets (PN):

- Authorisation Template (AT) at the design phase is defined where static parameters of the authorization are listed.

- when an activity starts execution, the AT is used to derive the actual authorisation value.

Static analysis is performed prior execution and all privileges that are granted during execution are stored with the WAM environment. However, it does not consider history of the flows. Further, it does not *monitor* the event(s) generated during the execution of task. Petri net was extended to what is known as Secure Petri Net (SPN) was proposed. SPN was used to detect conflicts and attempt to re-

[3]We note here that the support required in workflow security relates to the activity's granularity. This has motivated DAC's extension to cater for the security properties of Integrity and Authorization for workflow.

solve it. The ability to synchronize the flow of authorizations during the workflow execution is an important criterion when context changes as they do in modern workflow. For this reason, WAM is not adequate to support workflow security. Take for example the scenario of privileges which will be granted/revoked to/from the flow according to the events generated during the execution of an activity. In WAM the focus is only on the authorisation in an activity's state and primitives, but not in resource accesses. Whilst WAM can deal with Authorisation properties and that MLS handles Integrity properties within the task dependencies, neither can handle the security property of Availability. WAM was also extended in [67] and SecureFlow was proposed. Specifying authorisation constraints for role assignments was done using a simple 4GL language. No provision however is given to support different workflow security constraints from build-time to run-time of a workflow. Authorisation properties can be expressed but can not handle integrity and availability properties.

2.5 Workflow Adaptability

Central to context-aware workflow is their ability to *orchestrate* their activities and services according to user's needs. The ability to do so in heterogeneous environments poses a great challenge [89,105]. There are various approaches to deal with adaptability of workflows which can be classified as run-time, automated and

instance-based. In [135], a review focusing on the various adaptation approaches in context-aware workflows and has identified various characteristics shared by these approaches. Based on [61, 105], adaptation strategies for context-aware workflows are classified into *reactive* and *predictive* classes. The former occurs when an event has been triggered from the sensed context indicating the need for adaptation for the task(s) that is to be executed. This has a fundamental limitation as it does not take into account time-critical tasks. The later analyses early the effect of adaptation in advance. This in turns require the existence of a static adaptation plan which limits itself only to static environments. However, none of them deal with adaptation at a secure context and time-critical activities. Another important issue is that there is no workflow language available for the specification, modelling and designing context-aware workflow which takes into account security, timing and adaptability issues, [40, 134, 147]. Although context adaptation can be considered as the exception to the normal workflow and hence may be handled through database triggers and even-condition-action rules, [33, 147], our central philosophy here is that context-awareness, timing and security issues are the norm in workflow systems in real environments which are becoming more pervasive and complex. In what follows we review some of the early attempts in adaptation.

Existing Techniques. In [81], an extension to BPEL workflows is proposed to support policies to look up, select and bind partner services which are compliant with a port type that is defined within an activity in a workflow definition. Such a *find-and-bind* approach is not context-based neither compositional. In [37], BPEL was extended to allow adaptation of interfaces to human operators. The approach and its accompanied architecture (known as PerCollab) is still design-time, manual and evolutionary; it differs from approaches such as those in [5, 49] due to its purely vertical, refinement-based nature. Similarly, the work in [149] supports adaptation in declarative terms, focusing on various notions of inheritance: Four types of inheritance (*projection, protocol, projection/protocol, life-cycle*), are defined, corresponding to a set of rules to inheritance-preserving and transformation rules, and a verification process. Again, these notions of inheritance are rather vertical in nature. In [93], the authors adopt an approach based on the run-time modification of process instances by means of constraints on changes over code, which can be both of a mandatory and soft nature. Such constraints are expressed using a declarative language ConDec, whose semantics are given in a specification-oriented style where the underlying logic is a variant of linear temporal logic, and working on an automata representation of workflows. The approach, as such, is run-time and instance-based. In [94], a minimal set of rules is used in order to enforce changes in workflow instances. Such rules are based on the idea of refining the running code with specific instances. The design of

these rules are such that it is possible to formally guarantee that the adapted work-flows are persistently consistent and correct. The approach is not tailored to a specific work-flow language, but it is defined on a generic formal model of work-flow. In [125], the focus is focused on a suitable subset of the BPEL language. In both cases, the approach is run-time and instance-based. However, the approach lacks the ability to change policies according to change in contexts. In the WS-Diamond project, [1,12,13], the focus was to establish a methodology for defining *distributed self-healing* applications. With this aim, they focus on issues

- dealing with diagnosability and reparability testing of workflows; and
- , more relevant in our context, regards the enactment of self-healing mechanisms.

In this sense, an architecture was proposed which

- enriches the run-time support for the sake of diagnosing and fault repairing,
- in order to enact self-healing of process instances, and also
- to guarantee that certain QoS criteria are obeyed.

The work is rather at an infrastructural level, and concerns adapting single run-time workflow instances. In [48] an approach was given which relies on defining a set of test suit, for each service which is then used to monitor runtime deviations

from expected behaviours [50]. An architecture was proposed, if a deviation from an expected behaviour is discovered. The architecture tries to synthesize models (automatically) that approximates the interaction protocols, and steer client-side adaptations at runtime. The effectiveness of the approach is strongly dependent on

- the *exhaustiveness* of the test suite which is often provided by the user; and
- that the synthesis engine is rather *expressive* and hence its *performance* is acceptable.

This approach is categorized as run-time, instance-based, and (partially) automated. Predicting and learning the nature and types of faults given the history of a service usage (which is a classic engineering principle), in order to be able to repair them accordingly, is considered and dealt with in [116]. The ideas based on the observation that the process of the repair is often related to the type and the nature of the fault that caused the error. A repair strategy can be then constructed from analysing

- the temporal behaviour of the fault; and
- substitution is applied if the fault is transient but permanent, yet a retry is performed if the fault is transient but intermittent. The retry period will have to be calculated dynamically.

Learning strategies for fault repairs and the automatic selection an appropriate action for repairs is performed incrementally. The learning process are based on a Bayesian classification of faults; namely, permanent, intermittent and transient. The approach is fully automated, and operates on run-time instances of activities. The learning knowledge of repairs increases and indeed its decision making improves as more faults are repaired. In [56], the problem of run-time adaptation of a composition of services based on the evaluation of Quality of Service (QoS) in terms of expiration times was tackled. The approach is based on viewing processes as atomic entities, and considers adaptation as a reconfiguration of such entities. In [55], a similar approach is taken, considering a more general view of QoS in terms of information flow, but incurring higher overhead times when computing the runtime reconfiguration of services. In [36], a staged architecture that supports the adaptive composition and execution of services, named A-WSCE, is presented. The platform stands on top of the Synthy tool that allowed it to provide automated composition functionalities. The key idea in [36] is that of realizing a functionality by means of multiple, diverse compositions, and then selecting at run-time one of the available compositions, considering different QoS metrics and possibly replacing elements should run-time faults take place. The approach is embedded in an end-to-end approach that covers all the phases, from service composition to deployment and execution, and it is fully automated. Furthermore, in [106], the focus is on runtime adaptation based on

non-functional QOS measures. Specifically, the authors considered a composition of services, whose non-functional requirements can be adapted by modifying non-functional features of the component services. This is performed at run-time, based on an aspect-oriented approach that allows the decoupling of adaptation concerns with those of the business logics. There are approaches that, while still related to (built-in) adaptation, take different views and focus on different, and sometimes specific, issues. In the [91] approach, the focus is rather on guaranteeing the soundness, based on well-founded semantics, of adaptive workflows inside some given workflow management system. To achieve this, the authors inject workflows with semantic constraints of specific forms, facilitating the way to their formal verification, both considering run-time instances and adapted schemata of workflows. In [136], the focus is to guarantee the alignment between adaptable process schemata and their run-time instances. In particular, changes to a schema must reflect to the process instances, and because of misalignments, conflicts of various kinds may arise. The work identifies methods to identify such conflicts, based on execution equivalence and structural comparison. In [128], the idea is to support a specific form of *protocol* adaptation that takes place prior to deployment, and serves to adapt interacting services in such a way that the protocol constrains provided by a provider are obeyed by its users. This is performed by suitably abstracting away from the definition of protocol details prior to runtime, and then selecting an appropriate instantiation of actions at runtime, also based on a shared

semantics. This means that a fairly rich service description must be given, including roles, abstract and associated concrete protocol actions, and semantics. For this purpose, the authors devised a specific "Very Simple Choreography Language" (VSCL) language which embodies these elements. Similarly, in [29], an automated approach is proposed to analyze protocol mismatches, and generate adapters between services, expressed in BPEL. The whole process takes place at design time, and integrates an automated verification step to identify the presence of deadlock situations. As such, the approach does not lead to an adaptable code, but performs a unique adaptation prior to deploy time. In [104], a conceptually similar approach is taken, but using a more thorough analysis technique and therefore resolving, by means of a tree representing possible ordering mismatches, also deadlock cases.

2.6 Contexts: Modelling and Management

Fundamental to the current work is the fact that workflows are both security-critical and context-aware. We have thus far reviewed general principles of workflow systems, security requirements and what remains is to identify how modern workflow systems are context-aware. This section is devoted to this topic. The continual advances and growth in smart devices such as laptops, PDAs and smart phones with various processors and operating variety of sophisticated in-

frastructure,*mobile computing* has attracted variety of applications such as (modern) workflow systems. Activities within a workflow system may be executed on variety of these devices, each may be supported by different platforms. Activities can be executed and performed on, for example, one PDA in one instant but on a different device on another. Thus, in an attempt to move beyond the classic/-traditional view of explicit usage of computers and terminal devices, a new, yet general paradigm of *user-centric mobility* has been introduced in [146]. In this paradigm, smart and autonomous computing technologies are embedded in every device to enhance the use of computers and make them invisible to the users. In such paradigm, emphasis is on mobile data access, smart spaces and context awareness. Unlike tradition paradigms, where the majority of applications are transformational (in the sense that they rely on user input), the new paradigm relies on implicit information that can be deduced from (sometimes incomplete) description of the environment which in turns change rapidly. Hence "context" plays a vital role. Context in this rather ubiquitous environment refers to any information that can be used to identify and characterise an activity. Take for example a (traditional) home environments, the information about user's position is a context, and the times s/he is anywhere in the home are another contexts. These context information can be then used as transition constraint for an appropriate service selection and execution in workflows within ubiquitous environment [38]. In the literature, (e.g. [27,42,144,148], there have been different definitions of the

term context: Contexts can describe situations, Day [42] has elaborated on this view and said:

> *Any Information that can be used to characterise the situation of an entity. An entity is a person, a place, or a physical or computational object that is considered relevant to the interaction between a user and an application, including the user and application themselves.* [42]

Contexts can also be defined as *meta-information* to characterise the specific situation of an entity and to describe a group of conceptual entities [27]. However, the use of open-ended phrases such as "any information" or "characterise", render context to be so broad that it can cover everything. Winograd [148] has argued that there is a distinction between context and environment.

> *...context depends on the interpretation of the operations involved on an entity at a particular time and space rather than the inherent characteristics of the entity itself.* [148]

*C*ontext awareness is thus the ability for a software system to acquire, manage, interpret and respond to context and thus to provide appropriate services to the changing situation. In our view, a context instance is a set of values that describes

the states and relations of an entity at a specific point of time and which conform to a certain context type. For example, John, who is a Person, has context information that could be described as:

John: Person

Latitude ="45.87695”

Longitude = "-3.298875”

IsInvolvedIn = "Reading”

SitsBesides = "Alan”

One of the key issues with context awareness is how context information can be acquired from different sources (e.g. users, device, and environment), represented, managed and be integrated to be used adequately by a workflow applications.

Within workflow systems, context plays an important role. Adequate information about available contexts, the underlying WfMS is able to determine the most appropriate context to execute an activity. For example, knowing the contextual information about "John”, we may choose "Alan” to perform the activity instead of "John”.

2.7 Calculi for Context-Aware Systems

With the advent of context-aware systems, the need has grown to establish formal basis that underpins their development. As a result, various calculi have been

put forward and even a *Grand Challenge* initiative on Ubiquitous Computing, [4], have been established in the U.K. to address this need. Various attempts have been made to model workflow using the Activity diagram of UML [26] and variants of Petri net (for example see [142, 143] are used to provide a formal underpinnings. These variants extend the original Petri net with time, data and hierarchy. In [142], coloured tokens (to model data) and timing intervals associated to transitions (modelling duration of transitions) were added to the original Petri net. However a fundamental limitation of these attempts is their inability to model context and that workflows systems are treated as closed rather than open systems. Another formal modelling technique is Harel's Statemate Statecharts [52–54]. In Statechart however, data is updated by the system itself and not by the environment, and they share the same limitation as Petri net in dealing with context.

Milner has proposed the concept of *b*igraph, [96] which is a special type of graphs in which both communication amongst its node as well as their spatial configuration can be uniformally modelled. Nodes in a bigraph are permitted to be nested within each other to form nodes with more complex structure. Milner [78, 83, 96] has extended the bigraph model and introduced *b*igraphical reactive systems (BRSs) as a unifying framework for designing models of concurrent and mobile systems. In this model, reactive systems are represented as a set of rewriting rules together with an initial bigraph on which the rules operate. Plato-graphical models was introduced by Birkedal, et al., [25], as an attempt

to model context-aware systems. Their attempt to use BRS was concluded that BRSs are not suitable for directly modelling context queries. Context-awareness calculus was proposed by Zimmer [152]. Contexts in the calculus have a hierarchical structure similar to the one proposed in mobile ambients [32]. Further, the multi-agent synchronisation mechanism was adopted from the join-calculus. The Context-awareness calculus was further extended by Bucur and Nielson, in [30]. In this extension, ambients are able to publish context information with the context hierarchy. Context capabilities are modelled by named macros that capture primary context information. Further, in [47], a Calculus for Context Aware (CCA) was presented in which context are also structured and introduced the notion of context-guard which is used to manipulate and operate on context. Mobility is also modelled as ambient (similar to that in Mobile Ambient (MA) [32]). CCA is close to the work presented here, although our $\mathcal{CS}-\mathcal{F}low$ enjoys other features such as timing, schedulability, priorities and non-determinism. We shall return to CCA in Chapter(4) (on page 119).

Based on UNITY, Roman et. al. [126] presented another formal model, known as Context UNITY, in which aspects of context-aware computation can be expressed. In Context UNITY, context information are provided through *exposed variables* and existential quantification is used for context discovery. This is similar to our *frame* in $\mathcal{CS}-\mathcal{F}low$ (see Chapter(3). Furthermore, abstract data types were used to explicitly model contexts in CommUnity by Lopes and Fiadeiro [90].

They use four special observables, similar to exposed variables and *frame* found in [126], to provide context information on the entire system. Moreover, within a mobile university campus environment, Bouzeghoub et. al. [28] has proposed a model that performs situation-aware adaptive recommendation of information. Ontologically-based frameworks have also been used to model context, see for example [28, 39, 69, 118, 145].

2.8 Summary

In this chapter we gave a general overview of workflows and their importance in everyday businesses and enterprises. Often, workflows are conflated with business processes; throughout this thesis we take the view that both are identical unless we explicitly state the difference. There are two major concerns in current workflow system development. The first is **security** considerations and the second is **context awareness**. Modern workflow systems cross the boundaries of organisations, each has its own security requirements, policies and constraints. Even within one organisation, activities in a workflow systems may be executed, in one of its instances, within a platform but in another instance it may be executed or performed on a different platform with completely different environment. Indeed it may not even be automated. we have reviewed in some details Workflow Management Systems together with the associated Workflow Reference Model which is a de

facto standard produced by the Workflow Management Coalition (WfMC). We have also reviewed basic security requirements for workflow systems and it was clear that current Workflow Management Systems do not adequately deal with these requirements in the presence of contexts which change constantly.

In addition, within a highly dynamical environment, the notion of adaptability plays a central role in the design and implementation of workflow systems. There are various approaches to deal with adaptability of workflows which can be classified as run-time, automated and instance-based. However, none of them deal with adaptation at a secure context level.

Chapter 3

$\mathcal{CS}-\mathcal{F}low$

COMPUTATIONAL MODEL –

Linguistic Support

Objectives

- Describe the computational model for a context-aware and secure workflow systems.
- Details of the language $\mathcal{CS}-\mathcal{F}low$.

3.1 Introduction

As we have articulated in the last chapter, modern workflow systems are *highly distributed* and *cross the boundaries* of many enterprises. Take for example the situation in which we wish to purchase a product from Amazon.com. The purchase order follows a complicated workflow that crosses the boundaries of many organisations/enterprises such as

- Suppliers
- storage warehouses,
- Financial organisations,
- Transport/Haulage companies,
- Legal agencies,
- Insurance companies, etc.

Each will need to access information that, in many cases, are private: name, address, bank details, credit history, etc.

To deal with these security and privacy issues, the operations and codes of conduct in each of these enterprises should (and normally is) governed by set of regulations and policies that restrict access and limit the manipulation of their "trusted", "confidential" and "sensitive" information and/or local resources. In addition, within

each of these organisations, the activities may be executed/performed on different devices, agents and may be supported by different computational platforms/infrastructures.

Additionally, we have also discussed that workflows are intrinsically *context-aware* and should be able to interact seamlessly and unobtrusively with its surrounding. For example, in a warehouse situation, context requirement is *location* – it is required not in absolute values, but as relations between relevant humans and objects (for example, is a qualified worker (e.g strength) in the range of a box that needs to be transported?). Further, the interaction between humans and objects (for example, *picking up a crate* or *putting down a crate*) is equally important. Also, individual constraints of the objects (for example, *maximum transportable weight* and/or *special storage requirements*) of the involved entities have to be considered, etc.

Designers should therefore be able to define/specify constraints based on the context that the workflow perceives during its execution. This ability to change constraints/policies that characterise contexts, and hence security requirements, is important in order to increase our trustworthiness in the system.

This Chapter gives a detail description of the *Computation Model*, Section(3.2), for our Secure and Context-Aware Workflows. It also presents our design language $\mathcal{CS}-\mathcal{F}low$. The language has both graphical (Section (3.3)) and textual (Section(3.4)) representations. The former is brief with the sole purpose of clar-

ifying the ideas. The later presentation takes a usual and formal linguistic approach. The chapter ends with few examples of some of the common models, Section(3.5). The underpinning formal semantics of the language will be given in the next Chapter.

3.2 The Model

Fundamental to our design philosophy is that the computational model has *context* as a first class citizen which has a

- *name* to identify (model) its location and
- *frame* to identify all observable attributes of interest.

Another important consideration is the ability to express *security* (e.g. access control) and *context* requirements, in a unified fashion using the same mechanism and constructs.

It is also recognised that real-time is an important feature of workflow systems. From *deadlines*, and *worst case execution times* to *delays* and *interrupts*. We have carefully considered all these features and their associated specific constructs and have made the appropriate provisions. Furthermore, in dealing with real-time activities, our model should not make any of the usual simplified assumptions

which are commonly used with real-time formalisms. For example, the *maximal parallelism* hypothesis (i.e., the availability of an infinite number of resources), [129], and the instantaneous communication assumption [101, 132]. Further, the underlying design principles of any language must be

- richness, yet simplicity,
- practicality,
- support of modularity and, importantly
- formal underpinning (i.e. formal semantics).

To this end, our model has three distinct components: **Context**, **Activity** and **Guard**.

3.2.1 Context

As we mentioned earlier that context plays an important part in the operational behaviour(s) of modern workflows. Existing modelling techniques assume a centralised execution infrastructure. Therefore, one of the many challenges is to design workflow system to be executed in *pervasive* environments. This requires the provision for a computational model in which context is made first class citizen and that each of these contexts has its own security policies that constrain its behaviours - from access control to privacy and confidentiality provisions. Such a

computation model has to treat both constraints (security and context) in a uniform and integrated fashion.

Contexts can take a variety of forms: different platforms and operating systems, hand-held devices, web-services, etc. A context is characterised by, what we call *context frame*, which is a set of variables (or attributes) of interests. For example, attributes of interests of a context such as a PDA could be its *processor speed, memory size* and *battery life time*. On the other hand attributes such as *age, qualification* and *work experience* will be of interest in the case of a human context; yet *body temperature, blood pressure* and *kidney functions* are attributes more appropriate in the case of a hospital ward context.

In our model, these attributes are predicated upon to form a *context guard* – in the case of context, so as a decision may be taken to execute an activity or choose different but more suitable context, etc. In addition, they are also important as mechanisms to express security policies and for the design of variety of enforcement mechanisms of these policies that, for example controls access to sensitive data/information. As we shall see later, these guards play important role in the specification and design of context properties and security policy constraints.

An activity in our model does not exist in isolation. Indeed it requires a context to house it. Activities within a workflow move into a context to be executed but may choose to move out to another context in order to complete its functionality. In this way, context can be nested in a larger context in a compositional fashion.

In what follows, a detail of these components are given.

3.2.2 Activity

Our unit of computation in a workflow system is an *activity* which describes a piece of work that contributes toward the accomplishment of a given (functional and business) goal. Hence, an activity has

- *a goal*,
- *an input*,
- *an output*,
- *performed in a particular order* – e.g. in sequence/parallel/alternate with others,
- *associated with a particular context* – Contexts can be an organisation, a device, a service (e.g. web-based service) or a computational environment.,
- *uses resources/information*,
- *may affect more than one organisation unit*,
- *creates some value for users.* and
- *properly terminates* – in the same or in a different context.

Central to an activity is a an activity *frame*. The frame is a set of variables, known as **context** variables, which are allowed to change during the execution of the activity. The context variables characterises the attributes of interest about the context, e.g., *location, time, temperature, etc.*. Their changes are only *observed* and then acted upon.

Normally, an activity *starts, executes* and then *properly terminates* within the same context it started with. An activity starts in one context but may terminate in a different context. This means that an activity has the ability to be *mobile* and moves from one context to another. But as an activity in our model is tightly associated with a context, mobility occurred at a context level, i.e. an activity moves with its context. This is achieved by the execution of the context operation `to` α. (I.e. moving out of the current context and/or moving into a new context, α).

In our model, activities may be composed concurrently to produce a new activity which terminates if and only if all of its components terminate, i.e. we adopt the *distributed termination* convention. Further, without loss of generality, we assume a single clock for an instant of a workflow. Activities are also composed in alteration and in a non-deterministic fashions. An activity can also be conditionally executed after the passability of its condition or guard.

3.2.3 Guards

Each activity/context is governed by a set of *context* and/or *security* policies/constraints which are continually changing due to either the occurrence of an event and/or the passage of time.

Within workflow systems, *access control* policies play a fundamental role. Traditionally an access control policy is expressed in terms of (see, eg., [7, 17, 22, 73–76]):

- *subjects* – such as human, activities, platforms. These need to be authenticated before being allowed to access a resource (an object) – an object can not be modified unless access right is granted.

- *object*. This is a resource which is there to be used. It has a state where a subject can alter once it is granted to do so.

- *action* – is an activity where once the access is granted, it can be executed.

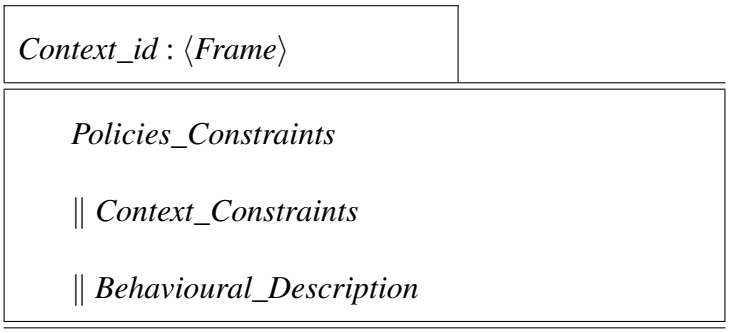
ECA is another formulation of policy which is discussed later on, see Section(3.5.7) (page 115).

When designing a policy we inevitably think of the system in a specific context and define the rules that restrict the system in this context. It is natural to define policies with respect to their context and then compose the policies to yield the overall system policy.

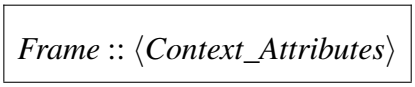
Security policies are expressed within the body of the activity and their enforcement can be part of the infrastructure of the activity itself or its context.

3.3 $\mathcal{CS}-\mathcal{F}low$: Graphical Representation

As a workflow is itself an activity, our presentation here makes no distinction between both. An activity exists within a context which has a *name*, a *frame* and an *activity* that specifies its functionality. In this section we give a graphical representation of our setting so as to increase clarity; however, a textual representation will also be given in a later section.



where *Frame* is given as:



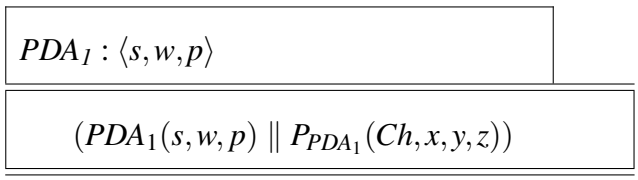
The classical notion of *frame* is that it contains variables that are allowed to change during execution time. *Context_Attributes* are the same but are read by all activities/workflows in the system, changed by the environment and not by the activities themselves. For example, sensors, clocks, etc.[1].

[1] An empty frame is denoted by ⟨ ⟩.

Note here that context and security requirements are constraints that are executed in parallel with the activity itself. The parallel construct here acts as the realisation of *enforcement*. Context and security requirements themselves are rules that are expressed as activities which constrain the functionalities of the workflow under design. we shall elaborate on this later in this chapter.

Take for example, an activity, such as *"Get Next Order"* or *"Track Progress"*, which may be executed on one of the *PDA_1* devices whose context parameters of interests might be: s (size), w (weight) and p (power_strength). The behaviours (context and functional descriptions) are expressed by the set of policies given by the processes P_{PDA_1} and PDA_1, respectively.)

This is represented graphically as follows:



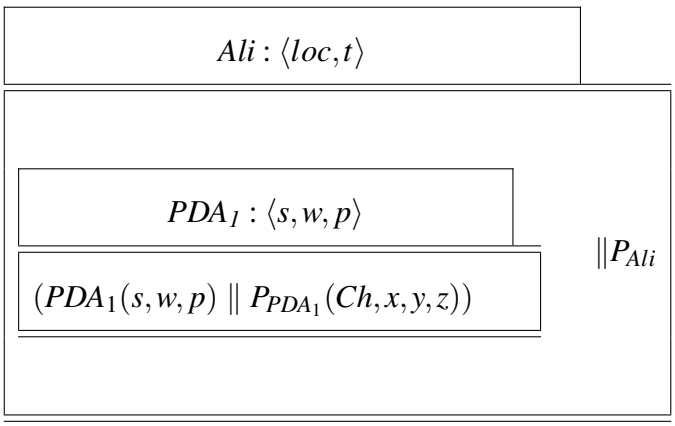
The notion of a behaviour here is interesting. Traditionally a *behaviour* is often defined as a sequence of *states*, as has been adopted within the logic-based formalisms, in particular, the temporal logic communities. However, in the process algebraic setting, such as CSP [62] or CCS [60, 97, 98, 100, 130], behaviours are formalised as a sequence of traces, specially *communication traces*. It all depends on what we can observe and if we are dealing with open or closed systems. In our case, being context-aware, it render itself to openness and so what we observe

are important. So we adopt trace-base semantics of behaviours. Therefore, we observe both types of traces: those which are constructed from context variables and other traces from activities' functional traces.

Within our setting, in which we have both states and context variables; both of which determine behaviours of the whole workflow, the definition of a behaviour may change.

Note that the state variables of interests are x, y, z.

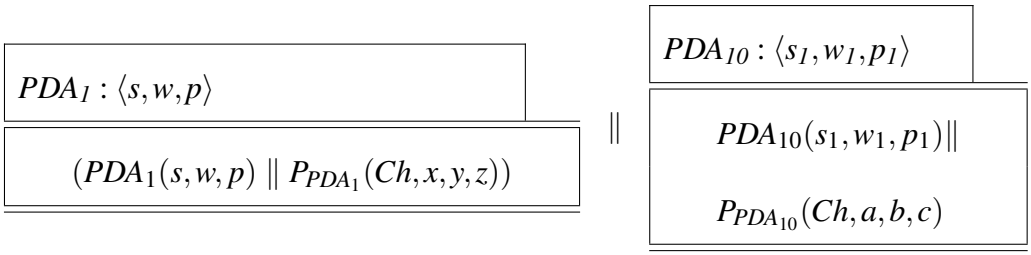
If we consider, the worker Ali, carrying this *PDA* may be modelled as a *parent* context, of which we are only interested in his location , l, and the time, t:



Here P_{Ali} is a process that describes the behaviour of Ali and whatever other devices he may be carrying with him.

At the same time, there may be some more activities that are being performed on

a different context, namely PDA_{10}. This is represented as



Note that $a,\ b,\ c,\ x,\ y,\ ,\ z$ and Ch are state variables describing the behaviour of an activity.

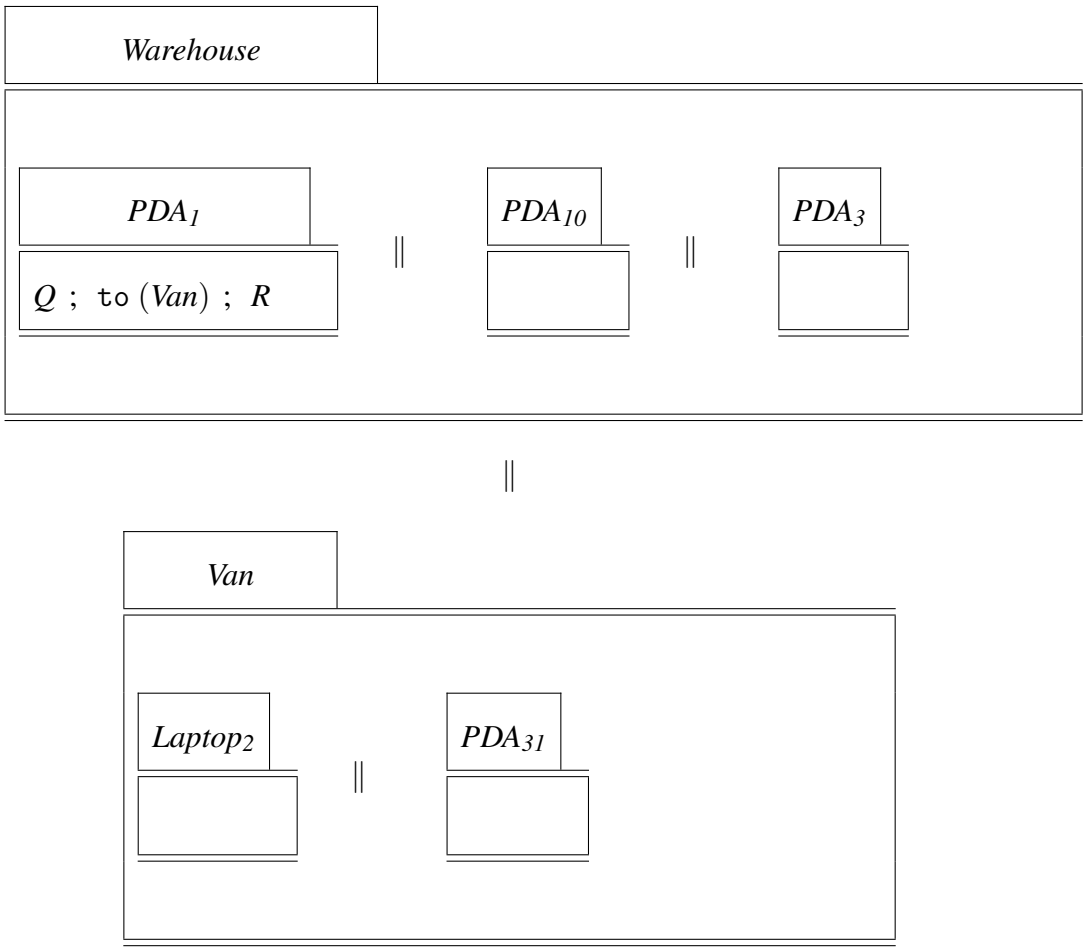
Activities can communicate by exchanging messages over channels. The communication is synchronous and is modelled using handshake message passing communication primitives: `C ! v` (output) and `C ? x` (input). For example, the processes P_{PDA_1} and $P_{PDA_{10}}$ may have the form: $P_{PDA_1} \mathrel{\widehat{=}}$ ...; `Ch ! Temp`$_{\texttt{value}}$; ...

and $P_{PDA_{10}} \mathrel{\widehat{=}} \cdots\ ;\ Ch\ ?\ x\ ;\ \cdots$. Here we have P_{PDA_1} is sending the value $Temp_{value}$ to $P_{PDA_{10}}$ over the channel Ch, which stores it in its local state variable, x.

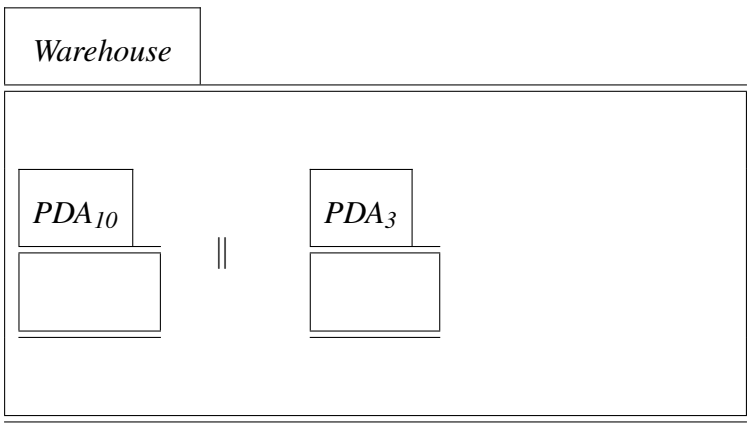
As we mentioned earlier, *mobility* can be also modelled. An activity can start in a context and then moves to be completed in another context using the operation `to` $(new\_Context)$.

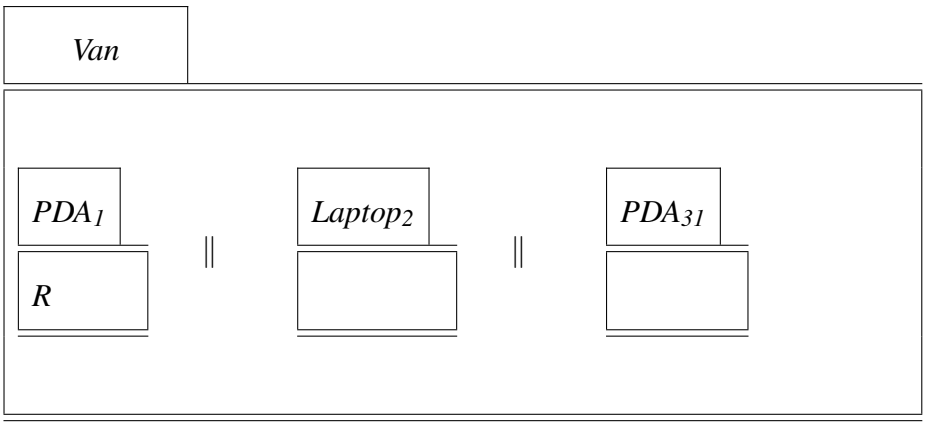
For example, in a typical warehouse, *Warehouse*, we may have three mobile devices inside its building: PDA_1, PDA_{10} and PDA_3. Also we may have a *Van* that is parked outside. On *PDA_1* the activity, $Q\ ;$ `to` $(Van)\ ;\ R$, starts with the process Q then followed by `to` (Van). This indicates that the activity has to complete (by

performing the process R in a different context *Van*. This is modelled as:



After an execution step, we have





It is important to note that PDA_1 still carries with it its side of channel Ch and may still continue to communicate with PDA_{10}.

3.4 $\mathcal{CS}-\mathcal{F}low$: Textual Representation

Our computational model is supported by a design language, known as Context,Secure-Flow ($\mathcal{CS}-\mathcal{F}low$), with which we are able to design and analyse workflows. Table 4.1 depicts the syntax of $\mathcal{CS}-\mathcal{F}low$ based on two syntactic categories: activities (denoted by P or Q) and guards (denoted by G_1 or G_2). We assume a countable-infinite set of names which are written in lower-case letters, e.g. x and y. Let `any` denotes an unspecified value. This is used in communicating signals amongst activities and/or context. We let b to denote Boolean variables and $\tilde{y}$ to

denote a list of names. As it is customary we use ˜ to denote a set, i.e., $\tilde{y}$ denotes a set of names where it is appropriate. We reserve t to represent time, such that $t \in \mathcal{N}$ and that t_α and t_β denotes the *start* and *termination* time of an activity, such that $t_\alpha \leq t_\beta$ I.e. the termination time of an activity will be at, or after its release time. We also use the names $\alpha, \beta, \gamma, \ldots$ to identify contexts.

Table 3.1 – Syntax of $\mathcal{CS}-\mathcal{F}low$

$$
\begin{array}{rcl}
P,Q & ::= & \texttt{skip} \mid \texttt{abort} \mid x := v \mid \texttt{delay}(t) \mid [t_1 \ldots t_n]\, P \mid c\,!\,v \mid c\,?\,x \\
& & \mid \alpha\,\langle \tilde{x} \rangle : \{P\} \mid \texttt{to}(\alpha) \mid \texttt{var}\,\tilde{x}\,\texttt{in}\,P\,\{Q\} \mid \texttt{chan}\,\tilde{c}\,\texttt{in}\,P\,\{Q\} \\
& & \mid \texttt{in}\,\alpha \cdot P(\tilde{x}) \mid P; Q \mid P \parallel Q \mid P \rhd_t^G Q \mid \texttt{while}\,G \cdot \texttt{do}\,P\,\texttt{od} \\
& & \mid [p_1] : G_1 \to P \,\Box\, [p_2] : G_2 \to Q \\
G & ::= & \texttt{true} \mid b \mid \texttt{not}\,G \mid G_1\,\texttt{and}\,G_2 \mid \texttt{somewhere}(\alpha) \cdot G
\end{array}
$$

The informal semantics of these constructs are as follows.

3.4.1 Contexts.

The context

$$
\begin{array}{l}
\alpha\,\langle \tilde{x} \rangle : \\
\quad \{ \\
\qquad P \\
\quad \}
\end{array}
$$

has a name, α, a frame $\langle \tilde{x} \rangle$ and its behaviour is described by the activity P. [2] For example, the context

$$
\begin{array}{l}
Ikea : \langle \quad \rangle \\
\quad \{ \\
\qquad P_{Ikea} \quad | \quad PDA_{23} : \{Q\} \\
\quad \}
\end{array}
$$

describes a context *Ikea* which has no frame, whose behaviour is described by P_{Ikea} and has a sub-context, PDA_{23}. The behaviour of this sub-activity is described by the process Q. A Context may have a *frame* that contains a set of variables of interests which can be captured, by existing sensors and sampled, then are acted upon. These are "read-only" variables. For example in *Ikea* warehouse,we may have $Damp_{Level}$ (monitoring the moisture in the air) and $Smoke_{Alarm}$ (monitoring fire). We can express this as:

$$
\begin{array}{l}
Ikea : \langle Damp_{Level}, Smoke_{Alarm} \rangle \\
\quad \{ \\
\qquad P_{Ikea} \quad | \quad PDA_{23} : \{Q\} \\
\quad \}
\end{array}
$$

The operation, $\texttt{to}(\beta)$ allows context to move out to another context, β. For example, *Ikea* above can take the form:

$Ikea : \langle Damp_{Level}, Smoke_{Alarm} \rangle$

[2] In process calculi, this is known as *restriction*, i.e. declaring variables that are only used by the activity/context.

$$
\begin{array}{l}
\{ \\
\quad P_{Ikea} \parallel PDA_{23} : \\
\qquad\qquad\qquad \{ \\
\qquad\qquad\qquad\quad \mathit{TakeStock}\ ; \\
\qquad\qquad\qquad\quad \texttt{not}\ (\mathit{Damp}_{Level} \geq 25 \vee \mathit{Smoke}_{Alarm}) \rightarrow \\
\qquad\qquad\qquad\qquad\qquad\qquad \texttt{to}(\mathit{Van})\ ;\ \mathit{Place\ Order} \\
\qquad\qquad\qquad \} \\
\}
\end{array}
$$

Unless $\mathit{Damp}_{Level} \geq 25$ or Smoke_{Alarm} has set off, the worker on PDA_{23} will place an order (i.e. perform $\mathit{PlaceOrder}$) for whatever stock(s) s/he has recorded (ie. after performing the workflow $\mathit{TakeStock}$). Otherwise, s/he has to complete stock taking before moving to the context known as *Van*.

Important to note the use of the BOOLEAN guard as a context expression. As we said earlier on that guards in $\mathcal{CS}-\mathcal{F}low$ play important role as they are used to constrain contexts as well as to describe security policies. We shall further elaborate on this in subsequent sections.

3.4.2 Activities.

There are primitives and compound activities.

Primitive Activities.

- $x := v$

  describes an activity that assigns the value v to a variable x. Once it is completed, it terminates immediately. In this setting we assume no consumption

of time during the assignment. If it does, we shall make this explicit using delays as appropriate.

- `delay`(t)

  is an activity that delays a workflow by a duration of t time.

- `skip`

  is the null activity. It is an activity that does not do anything and terminates immediately.

- `abort`

  is deadlock activity. It may be doing things but we can not observe it. It does not terminate. Such an activity should be avoided at all cost. In *refinement* calculi, rules are often formulated to avoid the refinement of the design to such activity .

- $[t_1 \cdots t_n]P$

  describes an activity P that will take either t_1, t_2, or t_n time units. If $n = 1$, the activity will take exactly a t time. [3] This is a powerful construct as it pro-

[3] Sometimes, we write $[S]P$, where S is a set $\{t_1 \cdots t_n\}$. Also as a shorthand we use:

$$\texttt{deadline}(n) \;\; P$$

to denote an activity with n as a specific deadline. This deadline representation is known as "hard" deadline, in the sense that the activity must successfully complete after n time unites. It is considered to be a timing failure if it completes either on $n-1$ or $n+1$ time unites.

vides a greater flexibility at execution time of a workflow. At run time, the scheduler (a component that is responsible for scheduling activities given various platforms and contexts) will have the choice between the various deadlines which will depend on the available resources.

- $c\,!\,v$ and $c\,?\,x$

  describes how activities are communicating with each others: $c\,!v$ is an activity that send the value v over the channel c; whilst $c\,?\,x$ describes an activity that receives a value over a channel c and stores it in x.

  $c\,!\,\texttt{any}$ and $c\,?\,\texttt{any}$ represent activities which exchange "signals", i.e the communicated value over channel c is irrelevant.

- $\texttt{var}\ \tilde{x}\ \texttt{in}\ P\ \{\ Q\ \}$ and $\texttt{chan}\ \tilde{c}\ \texttt{in}\ P\ \{\ Q\ \}$

  These activities, respectively, are used to declare/introduce set of state variables and set of channels used by the activity P and whose behaviour is described by the activity Q.

- $\alpha\ \langle\tilde{x}\rangle : \{P\}$

  This declares a context α which has a frame $\tilde{x}$, of observable attributes of interests, and whose behaviours are described by the activity P.

- $\texttt{to}(\alpha)$

  This models mobility. Upon execution, the current activity is moved to

another context α.

- $\texttt{in}\ \alpha \cdot P(\tilde{x})$

  This represents a "calling" activity: In the context α there is an activity, P, that is to be executed but with the "actual" parameters $\tilde{x}$. It is expected that within the context α there is an activity of the form $\texttt{var}\ \tilde{z}\ \texttt{in}\ P\ \{\cdot\}$. Here $\tilde{z}$ is a list of P's formal parameters. The body of P contains its implementation within its context. In a different context there may be different implementation. The implementation of P is completely hidden from the calling activity. In a process algebraic term, this represents process abstraction.

Compound Activities.

- $P\ ;\ Q$.

  It is an activity which starts with a sub-activity P and upon its proper termination, the sub-activity Q begins. Here we assume that the sequential composition is instantaneous. However, to model time consumption, we may utilise the delay activity $\texttt{delay}(t)$.

- $P \parallel Q$.

  This the parallel composition activity. The activities P and Q execute concurrently and the whole activity terminates if and only if both sub-activities terminate. In another words, we adopt distributed termination convention.

- `while` G `: do` P `od`.

  This describe an activity in which P repeats itself as long as G holds. G contains no communication activity.

- $P \rhd_t^G Q$[4].

  This is an activity that starts with the activity P. Then Q starts (from its initial state) if either a t time units has elapsed or an event has occurred (that made the guard G to evaluate to true) whichever happens first. The t and G are optional:

  - $P \rhd Q$.

    This is an activity which is equivalent to P.

  - $P \rhd^G Q$.

    This is an activity which starts with P and only if an event G occurs then Q takes over, i.e it acts as an interrupt.

  - $P \rhd_t Q$.

    Here the activity starts with P and after a t time units elapsed activity Q starts. Notice here that t is not related to the execution time of P. For example, if $t = 0$ then $P \rhd_0 Q$ is equivalent to Q. However, if t is greater than the execution time of P, then its termination has to be delayed until t elapsed, then Q starts.

[4]Sometimes, we use the term *signal* to refer to this construct.

- $[p_1]: G_1 \rightarrow P \,\square\, [p_2]: G_2 \rightarrow Q$.

  This activity is to model *choice*. The guards G_1 and G_2 are Boolean expressions which are evaluated at the start of the activity. p_1 and p_2 and integer values represents *priority*. We adopt the convention that p_1 is of higher priority than p_2 if $p_1 \langle p_2$. If both guards are evaluated to false, the whole activity fails and progress is not made until one of the guards is passable.

  If $p_1 = p_2$, then the activity models *non-deterministic choice*. In such a case, if both guards are evaluated to true, then one of the associated sub-activities is chosen at random and executed and then the whole activity properly terminates. None of the G_i contain communication activity. [5]

3.4.3 Guards

As we mention earlier, guards allow us to describe properties of contexts, similar to that adopted in CCA [47], as well as access control security policies' conditions. In addition to the usual propositional operators such as negation (`not`) and conjunction (`and`):

[5] The priorities here are fixed at design stage. However, we can enrich this prioritise choice by allowing priorities to change dynamically: For example, the activity $[p_1(i)]: G_1 \rightarrow P \,\square\, [p_2(i)]: G_2 \rightarrow Q$ changes its priorities according to the value of an integer variable i.

- `true` always holds.
- b Boolean variable.
- `somewhere`$(\alpha)\cdot G$ is a *spatial* modal operator that hold if there exists (at least one) a sub-context in which G holds. [6]

Some derived guards can be defined in Table(3.2).

Table 3.2 – Derived connectives

`false`	$\widehat{=}$	`not true`
b_1 `or` b_2	$\widehat{=}$	`not`(b_1) `and` `not`(b_2)
b_1 `implies` b_2	$\widehat{=}$	`not`(b_1) `or` b_2
b_1 `equivalent` b_2	$\widehat{=}$	$(b_1$ `implies` $b_2)$ `and` $(b_2$ `implies` $b_1)$

3.4.4 Postscript

In this postscript, we point out a number of important considerations that need to be adhered to in any $\mathcal{CS}-\mathcal{F}low$ design system. These are:

[6] Note that the duality of this guard, i.e. `everywhere`$(\alpha)\cdot G$ which holds if G hold everywhere within the context α can also be defined: `everywhere`$(\alpha)\cdot G \;\widehat{=}\;$ `not somewhere`$(\alpha)\cdot$`not` G

1. **Characteristics.**

   Our design of $\mathcal{CS}-\mathcal{F}low$ stems for the desire and the need for a workflow design language that has the provisions for modelling

   - concurrency;
   - real-time constraints;
   - context-awareness
   - mobility and
   - access control security policies

   in a uniform and modular fashion. $\mathcal{CS}-\mathcal{F}low$ achieves all of these unlike existing frameworks and languages. Process calculi such as CSP [62], CCS [95, 97, 99] and their variations are adequate to model concurrency, and some of their variations have the ability to model real-time systems [41, 124, 132]. However, they lack features to model context and security. The π-Calculus [97] and [130] and the recent *bigraph*s [25, 78, 83, 96] and CCA [47], were designed specifically for context and mobility but also lack real-time and security provision. Our recent development of S-Flow [10, 11] attempted to achieve this but context was not first class citizen and rather implicit. The language was complex as it has a sub-language that deals with security policies. The novelty of $\mathcal{CS}-\mathcal{F}low$ is that it has features

that allows the modelling of all of the above and that policies are elegantly expressed using guards which can also be used to express constraints on contexts.

2. **Design Faults.** Here we highlight few types of faults to avoid when using $\mathcal{CS}-\mathcal{F}low$ to design systems.

   (a) **The synchronisation Model.** The tight synchronisation model that we have adopted will require care in the design so as to avoid *communication deadlock* and other design faults. For example an activity such as

$$Ward : \langle light, temp \rangle \left\{ \begin{array}{l} \cdots \\ \| \cdots \\ \| \; c_1 \,!\, v \;;\; c \,?\, x \\ \| \; c \,!\, v_1 \;;\; c_1 \,?\, y \\ \| \cdots \\ \| \cdots \end{array} \right\} \tag{3.1}$$

   Obviously this design is fatally wrong as its realisation will lead to *deadlock*. This pattern may occur in various forms, the most difficult of which is if the communication appears in conditional such as the following:

$$\text{(3.2)} \qquad \mathit{Ward} : \langle \mathit{light}, \mathit{temp} \rangle \left\{ \begin{array}{l} \cdots \parallel \\ \cdots \\ G_1 \rightarrow P \,\Box\, G_2 \rightarrow Q \\ \cdots \parallel \\ G_{11} \rightarrow P_1 \,\Box\, G_{21} \rightarrow Q_1 \\ \cdots \\ \cdots \end{array} \right\}$$

where

$$P \mathrel{\widehat{=}} c \,!\, v$$

$$P_1 \mathrel{\widehat{=}} c_1 \,?\, x_1$$

$$Q \mathrel{\widehat{=}} c \,!\, v$$

$$Q_1 \mathrel{\widehat{=}} c_1 \,!\, x_1$$

This will be hard to analyse for it involves non-deterministic choice. Whilst we are discussing a design principle in which, at the analysis phase, we require to partition a given workflow into a sequence of *atomic* activities, a design such as the above will be fatally flaw and will not render to such analysis.

It should be noted that such a drawback is not in $\mathcal{CS}-\mathcal{F}low$ but rather inherent in synchronous communications.

(b) **Guards Passability.** In the construct, $[p_1] : G_1 \rightarrow P \,\Box\, [p_2] : G_2 \rightarrow Q$, when priorities are equal (i.e. $p_1 = p_2$, the choice becomes non-deterministic. This provides extra freedom in the design phase. However care must be taken to achieve (a) fairness and (b) avoid failure. Fairness in the sense that if both guards are passable every time, then

we should avoid choosing the same one. On the other hand, We should avoid a design that leads to the `abort` activity which will result in the case where both guards are not passable.

(c) **Scoping and Communication**. As we mentioned earlier that communication in $\mathcal{CS}-\mathcal{F}low$ is via message passing using channels and it is synchronous. The two communicating activities declare the channel(s) that are linked two. The sending activity pushes the communicated value in the channel and the receiving activity pulls the value and place it in a variable that is local to it. However, let us consider a model such as

$$\texttt{chan}\ c\ ,\ \texttt{var}\ x\ \texttt{in}\ \{\ c\ !\ v \parallel c\ ?\ x \parallel P(x)\ \} \tag{3.3}$$

This is a perfectly "legal" expression but a closer look reveals a flaw in the model as the variable declaration of x has a wider scope than intended. This will lead to undesirable behaviour. The intuition is that the receiving activity, receives the value in a variable local to it. This model can be rewritten as

$$\texttt{chan}\ c\ \texttt{in}\ \{\ c\ !\ v \parallel (\texttt{var}\ x\ \texttt{in}\{c\ ?\ x\ \}) \parallel P(x)\ \} \tag{3.4}$$

Here the x in $P(x)$ is different from the x used in the receiving activity.

3. **Context, Location** and **Holes**.

   Central to our model is that activities do not operate in the ether. They need contexts which identify their *locations* and within which they execute, terminate and may move out of them to another contexts. Unlike formalisms such as `CCA` [47], the notion of **holes** exists in which processes can move to. This makes the models rather clumsy and static with a fixed number of holes.

   The term "context" is used here instead of "location" for the later can indicate/require notions such as Proximity, Coordinates, Neighborhoods, etc. which in our view adds extra complication which is not needed.

   In our $\mathcal{CS}-\mathcal{F}low$, two special contexts, which we call *SKIP* and *STOP* The former is an empty context and nothing is happening in it and there are no observables. The later is the most un-inhabited context and will remain so forever! Further, if it moves into another context, it makes the host context un-inhabitable too. It is a context that needs to be avoided at all cost. Context, like activities, can communicate synchronously via channels. Whenever a context moves, its channels move with it. This is a powerful mobility notion as all what we needed is a single label to identify a context. The connectivity's between contexts (or their exact coordinates, neighborhoods, etc.) becomes irrelevant.

4. **Nondeterminism and Fairness**

   There are various types of nondeterminism:

   - *don't care* nondeterminism where the decision is made, when encountered, in an arbitrary way (unless *fairness* assumption is made. Correctness here is taken if every computation must succeed. Particularly, to be considered a terminating program if every computation (from an admissible initial state) of that program must terminate.

   - *don't know* nondeterminism, where correctness is defined existentially. Program succeeds if at least one of its computation terminates.

   In $\mathcal{CS}-\mathcal{F}low$ we adopt the former.The issue of fairness is beyond the scope of this study and is left for future work.

3.5 Examples

This section is devoted to provide few simple but illustrative examples of $\mathcal{CS}-\mathcal{F}low$ designs to some of the frequently occurring situations in workflows. From time-dependent issues to policies (both context- and security-related) structures. Some of these will be utilised in our evaluation case study.

3.5.1 Buffers

In this section we consider the design of buffers: one-place buffer that does not remember its contents once it is pulled out and another keeps a copy of the item that is deposited. The later is important in Context-aware applications.

1. One-place, memory-less Buffer. A one-place buffer is a classic data structure with two actions *push* (to place an item in the buffer) and *pull* (to remove an item from the buffer). Being a one-placed, the buffer is considered to be full when it contains one item and empty if it contains no item.
We assume that the buffer is empty to begin with (this a normal assumption). As it is customary with buffers, we have two operations: *push* (to place an item in the buffer) and *pull* (to remove an item from the buffer). The buffer behaves as follows:

- Initially, the *push* operation is performed. This is permitted as the buffer is empty.

- Alternate *pull* followed by a *push* are performed.

The application of the one-place buffer can be found in many situations, for example, an account in a banking scenario. From the informal requirements of the buffer, we can articulate a number of (security) policies governing each of the buffer's activities. These include

1. *removing an item from an empty buffer is not allowed*; and

2. *adding an item to a full buffer is not allowed*

And in a situation where a role-based scenario is important, we may add

3. *A buffer is initialised only by an authorised user.*

The system is modelled as follows.

$$
\begin{array}{lll}
(3.5) & \mathit{Buffer} \mathrel{\widehat{=}} & \left(\begin{array}{l}
\mathit{push}\,?\,x\,; \\
\mathit{empty} := \texttt{false}\,; \\
\texttt{while true} \\
\quad \texttt{do} \\
\qquad \{ \\
\qquad\quad \mathit{empty} \;\rightarrow\; \mathit{push}\,?\,x\,; \\
\qquad\qquad\qquad \mathit{empty} := \texttt{false} \\
\qquad\quad \Box \\
\qquad\quad \texttt{not}\ \mathit{empty} \;\rightarrow\; \mathit{pull}\,!\,v\,; \\
\qquad\qquad\qquad \mathit{empty} := \texttt{true} \\
\qquad \} \\
\quad \texttt{od}
\end{array}\right)
\end{array}
$$

and

$$
\begin{array}{lll}
\mathit{Authorised\_User} & \widehat{=} & \mathit{push}\,!\,v \\
\mathit{User} & \widehat{=} & \mathit{pull}\,?\,x \;;\; \mathit{push}\,!\,v
\end{array}
$$

2. Buffers with memory The above buffer does not remember an item once it is "pulled" out. Once it is taken out, the buffer does not have any record of it. In typical context-aware applications users date/preferences and device profiles, for example in Electronic Patient Records and smart devices. Data which are placed in the buffer need to be kept and be kept "fresh". Freshness here means being

updated. The amount of data depends on the size of the buffer. In this example, we still consider a one-place buffer but with memory. This type of buffer is sometimes known as *persistent Cell* [47]. Such a buffer/cell can be modelled as follows.

$$
\begin{array}{ll}
(3.6) & \textit{Buffer} \;\widehat{=}\; \left(\begin{array}{l}
\textit{Buffer} \;\widehat{=} \\
\qquad \begin{array}{l}
\textit{push} \,?\, x \,; \\
\textit{current} := x \,; \\
\textit{empty} := \texttt{false} \,; \\
\texttt{while true} \\
\quad \texttt{do} \\
\qquad \{ \\
\qquad\quad \begin{array}{ll}
\textit{empty} & \rightarrow \; \textit{push} \,?\, x \\
 & \qquad\quad \textit{current} := x \\
\multicolumn{2}{l}{\Box} \\
\texttt{not}\; \textit{empty} & \rightarrow \; \textit{pull} \,!\, \textit{current} \,; \\
 & \qquad\quad \textit{empty} := \texttt{true}
\end{array} \\
\qquad \} \\
\quad \texttt{od}
\end{array}
\end{array} \right)
\end{array}
$$

3.5.2 Adaptable activities

Consider a context-aware system which enables a mobile software agent to edit/view a text file on any host device using an appropriate text editor for that device's operating system. This can be modelled in $\mathcal{CS}-\mathcal{F}low$ using the concept of activity abstraction. Suppose there are two types of devices in the system: some running Windows operating systems and others running Linux. Each device running Windows is configured to use *notepad* as a default text editor while devices running Linux use *emacs*. Let's use *win* and *Linx* to denote context running Windows and Linux, respectively. Each of these contexts contains an activity ab-

straction *edit* that maps to the default text editor and the mobile agent just has to call that activity abstraction to edit a file using the local text editor as specified below. Our *ShopFloor* is specified as a context which has two devices: *win* and *linx*:

$$SpF \;\widehat{=}\; \left(\begin{array}{l} ShopFloor: \\ \{ \\ \quad win \parallel linx \\ \} \end{array} \right) \tag{3.7}$$

where the *win* context is specified as

$$win \;\widehat{=}\; \left(\begin{array}{l} win: \\ \{ \\ \quad \texttt{var}\, f\, \texttt{in}\, edit \\ \quad \{ \\ \quad\quad notepad(f) \\ \quad \} \\ \} \end{array} \right) \tag{3.8}$$

And the Linux device is specified as

$$linx \;\widehat{=}\; \left(\begin{array}{l} linx: \\ \{ \\ \quad \texttt{var}\, f\, \texttt{in}\, edit \\ \quad \{ \\ \quad\quad emacs(f) \\ \quad \} \\ \} \end{array} \right) \tag{3.9}$$

The employee/software agent may have the following specification

$$(3.10) \qquad Employee \;\widehat{=}\; \left(\begin{array}{l} Employee: \\ \quad \{ \\ \quad\quad \texttt{somewhere}\ (ShopFloor)\ \cdot\ edit(file) \\ \quad \} \end{array} \right)$$

If the host device is *win* then *notepad* will be used:

$$(3.11) \qquad win \;\widehat{=}\; \left(\begin{array}{l} win: \\ \quad \{ \\ \quad\quad \texttt{var}\ f\ \texttt{in}\ edit \\ \quad\quad \{ \\ \quad\quad\quad notepad(f) \\ \quad\quad \} \\ \\ \quad\quad \| \\ \\ \quad\quad Employee: \\ \\ \quad\quad\quad \{ \\ \quad\quad\quad\quad \texttt{somewhere}(ShopFloor) \cdot\ edit(file) \\ \quad\quad\quad \} \\ \quad \} \end{array} \right)$$

Otherwise, *emacs* will be used

$$(3.12) \qquad linx \;\widehat{=}\; \left(\begin{array}{l} linx: \\ \quad \{ \\ \quad\quad \texttt{var}\ f\ \texttt{in}\ edit \\ \quad\quad \{ \\ \quad\quad\quad emacs(f) \\ \quad\quad \} \\ \\ \quad\quad \| \\ \\ \quad\quad Employee: \\ \\ \quad\quad\quad \{ \\ \quad\quad\quad\quad \texttt{somewhere}(ShopFloor) \cdot\ edit(file) \\ \quad\quad\quad \} \\ \quad \} \end{array} \right)$$

3.5.3 Periodic Activities

In many occasions, workflows may contain several periodic activities. For example, in a context-aware hospital ward, patients are continually being monitored. Once a status of a patient has changed, a red light flashes indicating the need for measuring blood pressure at a regular interval and for a number of intervals. This can be modelled in $\mathcal{CS}-\mathcal{F}low$ as follows. Let T, D, and N denote a period, deadline and number of periods respectively, and assume $D \leq T$: Here we have a *Ward* workflow with two context variables, *light* (which can either be flashing or off) and *temp*. The following describes the situation, over a period of 12 hours, patients are admitted until the light flashes, then blood pressure is periodically monitored.

(3.13)

$$
\mathit{Ward} \;\widehat{=}\; \left(\begin{array}{l}
\mathit{Ward}: \langle \mathit{light}, \mathit{temp} \rangle \\
\{ \\
\quad \cdots \parallel \\
\quad \cdots \parallel \\
\quad \mathit{AdmitAPatient} \quad \rhd_{12}^{\texttt{somewhere}(\mathit{Ward}) \cdot (\mathit{light} = \mathit{flashing})} \quad \mathit{MeasureBP} \\
\quad \cdots \parallel \cdots \\
\}
\end{array} \right)
$$

where

$$
\text{(3.14)}\quad \texttt{MeasureBP} \mathrel{\widehat{=}} \left(\begin{array}{l}
\texttt{var } T,\ D,\ N,\ i \texttt{ in} \\
\quad \{ \\
\quad\ \texttt{while i} \leq \texttt{N} \\
\qquad\ \texttt{do} \\
\qquad\quad (\\
\qquad\quad ([T]([D]\mathit{TakeBloodPressure}) \\
\qquad\ \| \\
\qquad\quad \texttt{delay}(T) \\
\qquad\quad) \\
\qquad\quad i \ = \ i \ + \ 1 \\
\qquad\ \texttt{od} \\
\quad \}
\end{array}\right)
$$

where *TakeBloodPressure* is an activity that describes the flow of drug administration.

3.5.4 $\mathcal{ATM}$: Cash Withdrawal

To demonstrate the usage of guards, we consider the next generation of automatic teller machines, $\mathcal{ATM}$, whose purpose is to allow customers to perform the usual various financial transactions (e.g., *check balance, withdrawal of cash, change PIN, etc.*) but more **securely**. We model the $\mathcal{ATM}$ as a context which is composed of various sub-contexts, namely, *Transaction* (performing the actual transactions), *Camera* (which is integrated with the machine and automatically capture image of users once they are cleared), a database manager (*DBM*) that manages all

data (accounts, images, etc.) and an alarm, *Alarm* (which sounds when security is breached). These contexts need to communicate together via a pre-defined channels. Connections are assumed to exist, for example, between *Transaction* and the *DBM*, *Police* and *TakeImage* contexts. $\mathcal{ATM}$ has the following general structure:

(3.15)

$$\mathcal{ATM}: \langle : \rangle \left\{ \begin{array}{l} DBM : \langle ChipN, PINs, images \rangle \{P_{DBM}\} \\ \| \; Transactions : \langle names, ChipValid, PinNumber \rangle \{P_{Transactions}\} \\ \| \; TakeImage : \langle ChipN, images \rangle \{P_{TakeImage}\} \\ \| \; Police : \langle sound, images \rangle \{P_{Police}\} \\ \| \; Alarm : \langle bell \rangle \{P_{Alarm}\} \end{array} \right\}$$

Where P_X denotes a workflow that describes the behaviour of the context X. Here we only give a detail of the activity *Transactions*. Let *ReadChip* be an activity that enables the reading of the chip on a credit card and *CheckPinNumber* is another activity that checks the validity of the Personal Identification Number (PIN). An access control security role may be that a credit card must have a unique chip number and only a correct PIN, that is associated with that number can perform transactions.

(3.16)

$$
\begin{array}{l}
\textit{Transaction} \mathrel{\widehat{=}} \\
\quad \left(\begin{array}{l}
\texttt{chan } \textit{TDBM}, \textit{TCus} \texttt{ in} \\
\quad \left\{ \begin{array}{l}
\texttt{var } \textit{ChipN}, \textit{ChipValid}, \\
\qquad \textit{PinNumber}, \textit{ValidPin} \\
\texttt{while true} \\
\texttt{do} \\
\quad \textit{Continue} := \texttt{true}; \\
\quad \texttt{while } \textit{Continue} \\
\quad \texttt{do} \\
\qquad \textit{ReadChip}; \\
\qquad \textit{TDBM} \,!\, \textit{ChipN} \\
\qquad \textit{TDBM} \,?\, \textit{ChipValid} \\
\qquad \textit{ChipValid} \rightarrow \textit{Continue} := \texttt{true} \\
\quad \texttt{od} \\
\quad \textit{ValidPin} := \texttt{false}; \\
\quad \textit{counter} := 1; \\
\quad \texttt{while } \textit{counter} \leq 3 \texttt{ and not } \textit{ValidPin} \\
\quad \texttt{do} \\
\qquad \textit{TCus} \,?\, \textit{PinNumber} \,; \\
\qquad \textit{CheckPinNumber}(\textit{ValidPin}) \,; \\
\qquad \textit{ValidPin} \rightarrow \textit{PerformTransactions} \\
\qquad \Box \\
\qquad \texttt{not } \textit{ValidPin} \rightarrow \\
\qquad\quad \textit{counter} = \textit{counter} + 1 \\
\quad \texttt{od} \\
\texttt{od}
\end{array} \right\}
\end{array} \right)
\end{array}
$$

3.5.5 SWIFT-Cabs, Ltd.

SWIFT-Cabs, Ltd. is a local taxi firm whose headquarter (HQ) office is based in Leicester City Centre. It serves local community within the city as well as outside it including airports. Currently the firm has over 150 cars. This number is changeable as cars can leave the firm and new ones join. Cars are privately owned and once join the firm, an annual subscription fee has to be paid to the firm for its

service, namely allocating customers. Currently they operate using phone lines. Customers phone, or go to HQ, requesting a taxi. Customer are not aware of who the drivers are. The HQ is responsible for finding an available taxi.

This scenario is used widely within Service-Oriented computing paradigm and is known as Publish/subscribe protocol. It is an *asynchronous* messaging protocol. It is well documented that asynchronous mode of communication can easily be modelled by synchronous model by having a buffer between the sender and receiver. The buffer in this example is what is known as broker. In this scenario, senders (publishers) send messages only to the applications that are interested in receiving the messages (subscribers) without knowing their identities. The idea here is that such a decoupling between publishers and subscribers enhances and provides for greater scalability and a more dynamic network topology.

As we can see in our scenario of Swift-Cars Ltd: the HQ acts as the broker, subscribers are the cars and customers are the publishers. A simplified publish/-subscribe system can be modelled in $\mathcal{CS}-\mathcal{F}low$ as follows: The publisher, the broker and the subscribers can be modelled as three contexts named:
$Customer, HQ, Car_1, Car_{25}, Car_{33}$, respectively:

- *Customer* request a car from the *HQ*:

$$(3.17) \qquad Customer \mathrel{\widehat{=}} \left(\begin{array}{l} Customer : \{\texttt{while true} \\ \quad \texttt{do} \\ \quad\quad CustHQ\,!\,'pos' \,; \\ \quad\quad HQCust\,?\,x \\ \quad \texttt{od}\,\} \end{array} \right)$$

- The *HQ* receives the request, identifies available cars and forwards them the request:

$$(3.18) \qquad HQ \mathrel{\widehat{=}} \left(\begin{array}{l} HQ : \{\texttt{while true} \\ \quad \texttt{do} \\ \quad\quad CustHQ\,?\,req \,; \\ \quad\quad Identify\ \ appropriate\ \ cars \,; \\ \quad\quad (\, HQCar_1\,!\,'loc' \parallel HQCar_{33}\,!\,'loc' \parallel HQCar_{25}\,!\,'loc' \,) \,; \\ \quad\quad (\, \texttt{delay}\ (5) \parallel \\ \quad\quad\quad (\, \texttt{true} \rightarrow \triangleright_{Car_1HQ}\,?\,\texttt{any}\ HQCust\,!\,time \\ \quad\quad\quad \Box\ \texttt{true} \rightarrow \triangleright_{Car_{33}HQ}\,?\,\texttt{any}\ HQCust\,!\,time \\ \quad\quad\quad \Box\ \texttt{true} \rightarrow \triangleright_{Car_{25}HQ}\,?\,\texttt{any}\ HQCust\,!\,time) \\ \quad \texttt{od}\,\} \end{array} \right)$$

- The *Car* contexts have similar structure. Here we give the specification of Car_1:

$$(3.19) \qquad Car_1 \mathrel{\widehat{=}} \left(\begin{array}{l} Car_1 : \{\texttt{while true} \\ \quad \texttt{do} \\ \quad\quad HQCar_1\,?\,x \,; \\ \quad\quad Car_1HQ\,!\,\texttt{any} \\ \quad \texttt{od}\,\} \end{array} \right)$$

The whole system is modelled as

$$Swift-Cabs \;\widehat{=}\; \left(\begin{array}{ll} & Customer \\ \| & HQ \\ \| & Car_1 \\ \| & Car33 \\ \| & Car_{25} \end{array}\right) \tag{3.20}$$

It is worth noting that the above model can be generalised by parameterising $Swift-Cabs$ with an integer variable $NumCar$ which can be changed whenever a car is added to the system.

3.5.6 Ikea's shop floor

Let us consider the *Ikea*'s scenario, and let us assume that there are a number of smoke alarms, each is within a sub-context of *Ikea*'s shop-floor. One in the toilet, one in the children play area and another in the cashier area.

A desirable specification will be if all alarms have sounded then an evacuation workflow should be applied. However, if one of the alarms has sounded then this will cause the rest to sound and then the evacuation workflow is applied.

$$\begin{array}{l} Ikea : \langle Damp_{Level}, Smoke_{Alarm} \rangle \\ \quad \{ \\ \qquad P_{Ikea} \parallel PDA_{23} : \\ \qquad\qquad \{ \\ \qquad\qquad\quad TakeStock \\ \qquad\qquad\qquad [0]\ \texttt{everywhere(Ikea)}.Smoke_{Alarm} \rightarrow Evac \end{array}$$

$$
\begin{array}{l}
\quad \square \\
\quad [1]\ \texttt{Somewhere}(\texttt{Ikea}).\mathit{Smoke}_{\mathit{Alarm}} \rightarrow \\
\qquad\qquad (\mathit{RaiseAllAlarms}\ ;\ \mathit{Evac}) \\
\quad \square \\
\quad [2]\, \mathit{Damp}_{\mathit{Level}} \geq 25 \rightarrow \mathit{ResitPressure} \\
\mathit{PlaceOrder} \\
\}
\end{array}
$$

$\}$

3.5.7 Policies

As we mentioned earlier that policies are expressed using the choice construct where, in rule-based system, guards are used to express constraints (*premise* to, e.g. control access), to an action (which is the corresponding activity). However, in the *event-condition-action* (ECA in short) model, rules are used to model the behaviours of context-aware applications. The general form of an ECA rule is:

`ON` *event* `IF` *condition* `DO` *action*

where

- *event* signals a change in the external environment,
- *condition* is a Boolean expression about the state of the system and
- *action* is a computation of some kind.

This rules simply says that when an even occurs and if the condition is passable then the action is taken. In $\mathcal{CS}-\mathcal{F}low$, the general structure of *ECA* may take the

form:

$$
(3.21)\quad ECA \;\widehat{=}\; \left(\begin{array}{l}
\texttt{while true} \\
\texttt{do} \\
\quad \{ \\
\qquad\qquad G_{event_1}\ \texttt{and}\ G_{condition_1} \;\rightarrow\; P \\
\qquad\qquad \Box \\
\qquad\qquad G_{event_2}\ \texttt{and}\ G_{condition_2} \;\rightarrow\; Q \\
\qquad\qquad \cdots \\
\qquad\qquad \Box \\
\quad \} \\
\texttt{od}
\end{array}\right)
$$

Where

- G_{event_i} is a context-expression about the external context of that activity;
- $G_{condition_i}$ is a guard expression about that activity; and
- $P,\ Q$ are activities.

The whole workflow system may have the general form:

$$System \;\widehat{=}$$

$$Flows \parallel EventAnalyser \| ECA$$

Flows describe one or more activities that form the workflow in hand. *EventAnalyser*, receives an event and identifies the corresponding set of policies that need to be

executed [7]. *EventAnalyser* plays the role of Reference Monitor [11,73,75,76] that often used to enforce policies.

3.6 Summary

This chapter starts from the realisation that modern workflow systems are (a) highly distributed, (b) cross the boundaries of many enterprises. This makes current workflow systems

- **security-critical** – as it may need to access data of various enterprises;
- **context-aware** – as activities may be adapted to conform with the context that it occupies and
- some of their activities are inherently temporal with timing constraints.

In this chapter we have motivated the need for a novel design notation for workflow systems. Therefore any new notation/language has to enjoy the following features:

- support concurrency;
- context and context awareness are first-class citizen;

[7] Construct, such as, the interrupt, $P \triangleright_t^G Q$, can be utilised. Detail case study will be given in subsequent Chapter.

- supports mobility as activities can move from one context to another;
- has the ability to express timing constrains: delay, deadlines, priority and schedulability;
- allows the expressibility of (*access control*) security policies without the need for an extra linguistic complexities; and
- enjoy sound formal semantics that allows us to animate design and compare various designs.

Further, current specification and design languages for these systems are not adequate. As a response to these challenges, the chapter gives a novel computational model for *context* and *secure* workflow systems. The unit of computation is an *activity* which is a discrete unit, with input, output and satisfies a given business goal. An activity needs a context to execute. This makes context first class citizen. Context has a frame which has a set of observable attributes of interest: time, location, temperature, etc. We have introduced a design language ($\mathcal{CS}-\mathcal{F}low$) within which context and activity can be modelled. Constraints on contexts and security policies can be expressed in a unified manner. However, it is important to realise that activities can be **long-running** transactions and that they should be designed to tolerate failures, whether, timing or functional failure. Issues such as dealing with partial execution of a workflow before a fault occurs must be dealt with at a design time. This issue will be treated in chapter 5.

Chapter 4

$\mathcal{CS}-\mathcal{F}low$

FORMAL SEMANTICS

and Algebraic Characterisation

Objectives

- Provide a formal semantics of $\mathcal{CS}-\mathcal{F}low$ in `CCA`
- Give equational laws of $\mathcal{CS}-\mathcal{F}low$
- Present a timing characterisation of $\mathcal{CS}-\mathcal{F}low$

4.1 Introduction

In order to add rigor to our $\mathcal{CS}-\mathcal{F}low$ design language, we need to precisely define its formal semantics. This will add more confidence in the design and its use by highlighting any errors and/or limitations.

The techniques of presenting a semantic definition of any language can be categorised under three major headings:

- *Denotational* approach [123,131,133,137,139,140] which assign each construct/notation of the language to a value in a mathematical domain which is understood independently such as, for example, a function or logical formula that describes some kind of observation of the properties and behaviour of program when executed .

- *Algebraic* [31,59,60,108,109,115] style which does not explicitly say what the program does but it can show, using equational reasoning, if two differently written programs are equivalent or not.

- *Operational* style [51, 100, 120] describes how a program can be executed by a series of structured steps by some abstract mathematical machine.

The quest for the type of semantics we chose, we are guided (and minded) by the requirements/desire to animate and analyse our designs. The type of semantics we chose to opted for is what is known as *specification-oriented* semantics, which

is denotational in nature. In such semantics, each construct, in $\mathcal{CS}-\mathcal{F}low$, is described behaviourally in an abstract and formal language (we call this the *base* formalism). Normally, *specification-oriented* semantics has its root in logic as its "formal" base formalism. However, as long as the base formalism is sufficiently abstract, it is not necessarily to be a logic-based. The behavioural description could be done using, for example, traces [62, 63],a process algebraic style [31, 59, 60, 108, 109, 115] or be indeed a Petri-net style [119].

We have to be cautious here and re-iterate Hoare's account [63] of the origin of the terminology. Denotational semantics of a programming/design/specification language, makes the distinction clear between the syntax and the semantics. The later was wholly presented within the mathematical domain of partial functions. These characteristics have been acknowledged and regarded as definitive of the nature of denotational semantics. What we call "denotational" has been termed *predictive* [57, 58], *specification-oriented* [64], *application-oriented*, *observational* or even *relational* [16, 92]. The pioneer of the direct style of denotational is Mosses [103] and was justified by its original significance [137, 138]:

- each construct/component of the program has a meaning which is independent of its text of the manner of its execution, and
- the meaning of a larger program can be derived as a mathematical function of the meaning of its syntactic constituent, not of their syntactic form.

Given our desire to animate our $\mathcal{CS}-\mathcal{F}low$ specifications/designs, expressing the semantics in an executable base formalism is important.

However, our basic requirements is that the base formalism is sufficiently rich to be able to express

- context and context-awareness,
- mobility,
- real-time constructs,
- concurrency and
- supports animation and (formal) testing.

In the π-Calculus [98], [100], [60], [97], [99], and [130] or Duration Calculus [35] (and their derivatives), expressing context and timing constraints are hard and sometimes impossible. Bigraphs [25, 78, 83, 96] and the Ambient Calculus [32] are suitable for context, concurrency and mobility but lack in timing constraints and executability.

The Calculus for Context-aware Ambient (`CCA`) [47], which is based on the π-Calculus ([98], [60], [100], [97], [99], [130]) and borrows aspects of Bigraphs [25, 78, 83, 96], is sufficiently abstract and support all of the above but difficult in dealing with real-time issues. For these reasons we have chosen `CCA`.

In addition, in this chapter, we also explore algebraic characterisation of $\mathcal{CS}-\mathcal{F}low$. In doing so, we established a number of algebraic laws which can be used as

- a basis for building equational theory for $\mathcal{CS}-\mathcal{F}low$, and
- a tool to manipulate $\mathcal{CS}-\mathcal{F}low$ models with the aim of adding, e.g., efficiency to a given design.

The Chapter is therefore organised as follows. For the sake of completeness, in Section 4.2 we briefly introduce CCA. We refer the readers to original article for more details.

The formal semantics of $\mathcal{CS}-\mathcal{F}low$ given in CCA is also given in Section(4.3). Algebraic characterisation of $\mathcal{CS}-\mathcal{F}low$ is given in Section(4.5) whilst its timing characterisation and proof rules are given in Section(4.6).

4.2 Calculus of Context-aware Ambient – CCA [47]

We begin by giving a brief introduction to CCA. This presentation is largely based on the original work in [47].

CCA is a process algebra which is designed for modelling and analysing context-aware systems. An executable version of the calculus is also available to assess in models' animation. There are two main features of the calculus, namely *mobility* and *context-awareness*. The central notion in CCA is that of *ambient*. An ambient

represents an abstraction of a place (physical, logical, mobile or immobile– as well as the environment) which can be mobile and move in and out of another ambients forming a hierarchical tree structure where an ambient may be contained within another ambients – forming notions such as *parent, children* and *sibling* ambients. In addition to children ambients, an ambient can also contain a "process" specifying its capabilities (capabilities are used to describe behaviours). Capabilities in CCA are classified into *mobility, context-aware* and *communication* capabilities.

This section presents the syntax and the informal semantics of CCA. Due to the space limit, only features relevant to our work are presented. We refer interested readers to [47] for the full details of the calculus.

Table 4.1 depicts the syntax of CCA, based on three syntactic categories: processes (denoted by P or Q), capabilities (denoted by M) and context-expressions (denoted by E)[1].

Following the tradition of process calculi, e.g. in the π-calculus ([97], [100], [98], [130]), *names* are assumed to be the simplest entities in the calculus. We also assume a countably-infinite set of names, e.g., n, x and y. We let $\tilde{y}$ to denote a list of names and $len(\tilde{y})$ to denote the arity of such a list. $\tilde{y}$ is used sometimes as a set of names where it is appropriate.

Next, we enlist the informal semantics, [47], of these constructs:

The process $\mathbf{0}$, does nothing and terminates immediately. The process $P \mid Q$

[1]Context-expressions describe properties of contexts.

$$
\begin{array}{rcl}
P,Q & ::= & \mathbf{0} \mid P \mid Q \mid (\nu\, n)\, P \mid !P \mid n[P] \mid \{P\} \mid E?M.P \\
 & & \mid \texttt{find}\, \tilde{x} : E \,\texttt{for}\, P \\
M & ::= & \texttt{in}\, n \mid \texttt{out} \mid \mid \alpha\, \texttt{recv}(\tilde{y}) \mid \alpha\, \texttt{send}(\tilde{y}) \mid \texttt{del}\, n \\
\alpha & ::= & \uparrow \mid n\uparrow \mid \downarrow \mid n\downarrow \mid :: \mid n :: \mid \varepsilon \\
E & ::= & \texttt{true} \mid \bullet \mid n = m \mid \neg E \mid E_1 \mid E_2 \mid E_1 \wedge E_2 \mid \\
 & & \oplus E \mid \diamond E
\end{array}
$$

Table 4.1 – Syntax of CCA ([47])

denotes the process P and the process Q running in parallel. The process $(\nu\, n)\, P$ states that the scope of the name n is limited to the process P. The replication $!P$ denotes a process which can always create a new copy of P (i.e. $!P$ is equivalent to $P \mid !P$). The process $n[P]$ denotes an ambient named n whose behaviours are described by the process P. A context expression $E?M.P$ is a process that waits until the environment satisfies the context expression E, then performs the capability M and continues like the process P. The dot symbol '.' denotes the sequential composition of processes. $M.P$ denote the process $\texttt{true}?M.P$, where `true` is a context expression satisfied by all contexts.

The capabilities $\alpha\, \texttt{send}(\tilde{z})$ and $\alpha\, \texttt{recv}(\tilde{y})$ are used to send and receive a list of names from a location α. The location α can be '$\uparrow$' for any parent, '$n\uparrow$' for a specific parent n, '$\downarrow$' for any child, '$n\downarrow$' for a specific child n, '::' for any sibling, 'n ::' for a specific sibling n, or ε (empty string) for the executing ambient itself.

The capability `del` n deletes an ambient of the form $n[\mathbf{0}]$ situated at the same level as that capability (i.e. the process $\texttt{del}\, n.P \mid n[\mathbf{0}]$ [2] reduces to P).

[2] An ambient that is contains the $\mathbf{0}$ process is denoted by $n[\mathbf{0}]$.

4.3 Process Algebraic-Style Semantics of $\mathcal{CS}-\mathcal{F}low$

In this section we give an process algebraic style semantics for $\mathcal{CS}-\mathcal{F}low$. The base formalism is CCA. The semantics is to model each $\mathcal{CS}-\mathcal{F}low$ activity in CCA. But we begin by dealing with variables and channels.

4.3.1 Variables

A variable x is a memory cell and is modelled by the following process abstract where v is the initial value of the variable x:

$$mem \triangleright (x,v).\ x \left[\begin{array}{lll} \langle v\rangle.\mathbf{0} & & | \\ \uparrow().(w).\{\uparrow\langle w\rangle.\mathbf{0} & | \quad \langle w\rangle.\mathbf{0}\} & | \\ \uparrow(u).(y).\{\uparrow\langle\rangle.\mathbf{0} & | \quad \langle u\rangle.\mathbf{0}\} & \end{array}\right]$$

Read a value of a variable x into the the name n is modelled as the process:

$$x\downarrow().x\downarrow(n)$$

Similarly, writing a value t in a variable x can be modelled by the process

$$x\downarrow\langle t\rangle.x\downarrow\langle\rangle$$

4.3.2 Channels

A channel c is modelled in CCA as an ambient of name c. Communication over channels is modelled below.

4.3.3 Primitive Activities

$$[\![\texttt{skip}]\!] \mathrel{\widehat{=}} \mathbf{0}$$

$$[\![\texttt{skip; P}]\!] \mathrel{\widehat{=}} [\![P]\!]$$

$$[\![x := v;\ P]\!] \mathrel{\widehat{=}} x{\downarrow}\langle v\rangle.x{\downarrow}().[\![P]\!]$$

$$[\![c!v;\ P]\!] \mathrel{\widehat{=}} c[\uparrow\langle v\rangle.0]\ .\ [\![P]\!]$$

$$[\![c?v;\ P]\!] \mathrel{\widehat{=}} c{\downarrow}(v).\texttt{del}\ c.[\![P]\!]$$

$$[\![\alpha : \{P\}]\!] \mathrel{\widehat{=}} \alpha[P]$$

$$[\![\texttt{to}(\alpha);\ P]\!] \mathrel{\widehat{=}} (\nu\ m) \left\{ \begin{array}{ll} m[\uparrow\langle\rangle.\mathbf{0}] & \mid \\ (at(\alpha))?m{\downarrow}().\texttt{del}\ m.\texttt{out}.[\![P]\!] & \mid \\ \texttt{in}\ \alpha.m{\downarrow}().\texttt{del}\ m.[\![P]\!] & \end{array} \right\}$$

$$[\![\texttt{var}\ x\ \texttt{in}\ P]\!] \mathrel{\widehat{=}} (\nu\ x)\ \{[\![P]\!]\ \mid\ mem\langle x,0\rangle\}$$

$$[\![\texttt{chan}\ c\ \texttt{in}\ P]\!] \mathrel{\widehat{=}} (\nu\ c)\ [\![P]\!]$$

$$[\![[T_1,\ T_2,\ \cdots,\ T_n]\ A]\!] \mathrel{\widehat{=}} (t_\beta\ -\ t_\alpha)\ \in \{T_1,\ T_2,\ \cdots,\ T_n\}\ \wedge\ [\![A]\!]$$

Note that t_α and t_β are special integer variables denoting the start and end time of an activity.

4.3.4 Compound Activities

$$[\![P \parallel Q]\!] \mathrel{\widehat{=}} [\![P]\!] \mid [\![Q]\!]$$

$$[\![\texttt{while}\ G.P]\!] \mathrel{\widehat{=}} (\nu\, x)\left\{\begin{array}{l} x[\uparrow\langle\rangle.\mathbf{0}] \mid \\ (\neg[\![G]\!])?x\downarrow().\texttt{del}\ x.\mathbf{0} \mid \\ !\,([\![G]\!])?x\downarrow().\texttt{del}\ x.[\![P]\!].x[\uparrow\langle\rangle.\mathbf{0}] \end{array}\right\}$$

$$[\![p_1 : G_1 \to P \,\square\, p_2 : G_2 \to Q]\!] \mathrel{\widehat{=}} (\nu\, x)\left\{\begin{array}{l} x[\uparrow\langle\rangle.\mathbf{0}] \mid \\ (([\![G_1]\!] \wedge \neg[\![G_2]\!]) \vee ([\![G_1]\!] \wedge [\![G_2]\!] \wedge p_1 \geq p_2)) \\ \quad ?x\downarrow().\texttt{del}\ x.[\![P]\!] \mid \\ (([\![G_2]\!] \wedge \neg[\![G_1]\!]) \vee ([\![G_2]\!] \wedge [\![G_1]\!] \wedge p_2 \geq p_1)) \\ \quad ?x\downarrow().\texttt{del}\ x.[\![Q]\!] \end{array}\right\}$$

$$[\![\texttt{delay}(t);\, P]\!] \mathrel{\widehat{=}} (\nu\, clock)\ (\nu\, m)\left\{\begin{array}{l} clock[\langle 0\rangle.\mathbf{0} \mid\ !\uparrow().(w).\{\uparrow\langle w\rangle.\mathbf{0} \mid\ \langle w+1\rangle.\mathbf{0}\}] \mid \\ m[\uparrow\langle\rangle.\mathbf{0}] \mid \\ !\, m\downarrow().clock\downarrow().clock\downarrow(x).\{(x=t)?\texttt{del}\ m.[\![P]\!] \mid \\ \neg(x=t)?\texttt{del}\ m.m[\uparrow\langle\rangle.\mathbf{0}]\} \end{array}\right\}$$

4.3.5 Semantics of guards

$[\![\texttt{true}]\!] \mathrel{\widehat{=}} \texttt{true}$

$[\![\neg G]\!] \mathrel{\widehat{=}} \neg[\![G]\!]$

$[\![G_1 \wedge G_2]\!] \mathrel{\widehat{=}} [\![G_1]\!] \wedge [\![G_2]\!]$

$[\![G_1 \vee G_2]\!] \mathrel{\widehat{=}} [\![G_1]\!] \vee [\![G_2]\!]$

$[\![\texttt{somewhere}(\alpha).G]\!] \mathrel{\widehat{=}} \Diamond\, \alpha[[\![G]\!]]$

4.4 CCA Versus $\mathcal{CS}-\mathcal{F}low$

In this section we shall outline the important differences between CCA, and $\mathcal{CS}-\mathcal{F}low$ which serves as rational for the reason CCA was our choice.

- *v*ariables. CCA has no notion of variables which makes it more abstract but that makes it rather difficult to model. This is because variables (e.g. channels have to be modelled as ambients and have a particular structure. On the other hand, in $\mathcal{CS}-\mathcal{F}low$ variables and channels can be easily declared and hence behaviours of activities can be determined.

- *C*ommunication. Communication is $\mathcal{CS}-\mathcal{F}low$ is synchronous. Two communicating activities/contexts are linked via a channel (at modelling time, this is taken to be logical). With this, mobility is much more easier and where ever an activity moves, it channel(s) move with it. This is not the case in CCA.

- *R*eal-time Provision. Unlike CCA, $\mathcal{CS}-\mathcal{F}low$ have a rich constructs to model real-time activities. From delays, interrupt to deadlines which help in schedulability. This is an important characteristic as activities in workflow can be time dependent.

- *C*ontext. In CCA context are hierarchical and identifying neighbors is important. For example, we have concepts such as parent, children and siblings. In workflows this structure are not needed In $\mathcal{CS}-\mathcal{F}low$, contexts are flat and their identities and relationship is not needed, In $\mathcal{CS}-\mathcal{F}low$ we utilise the context's name to manage the complexity of context structure.

4.5 Algebraic Characterisation of $\mathcal{CS}-\mathcal{F}low$

In this section we give a characterisation of $\mathcal{CS}-\mathcal{F}low$ with some useful algebraic laws for $\mathcal{CS}-\mathcal{F}low$. These laws serve as a useful tool to manipulate a design in $\mathcal{CS}-\mathcal{F}low$ and a mechanism to characterise its semantics.

The algebraic laws allow us to give a precise and succinct description of each of the $\mathcal{CS}-\mathcal{F}low$ operator. Yet, the laws arise often from our informal understanding of $\mathcal{CS}-\mathcal{F}low$ constructs work. As we can see, some of these laws are similar to their correspondence laws in other process calculi, e.g. CSP and CCS [62], [63], [98]. An important issue that needs to be mentioned is that these laws are indeed all congruences in the language's denotational semantics given in Section(4.3). We have not considered the construction of *normal form* for $\mathcal{CS}-\mathcal{F}low$. This was considered outside the scope of the thesis. However, it will be interesting to develop such a form even if for a small and restricted subset of $\mathcal{CS}-\mathcal{F}low$ (for example, "finite" models/designs).

Denotational semantics of [57] map each activity into a domain with a partial order according to which one activity is greater than another if it is better defined, or more predictable. Unless indicated explicitly, we make no formal distinction between the text of a program and its value (semantics). If P and Q are activities, we will write $P \sqsubseteq Q$ when the semantic value of P is less than that of Q.

Definition 4.5.0.1 *An activity P is* refined *by an activity Q, denoted* $P \sqsubseteq Q$*, if Q performs as well as P, or better for any purpose.*

This can be expressed as

$$P \sqsubseteq Q \quad \widehat{=} \quad (G_1 \rightarrow P \,\square\, G_2 \rightarrow Q) = P$$

The following algebraic laws, using the structural congruence relation $\equiv$, allow the manipulation of the structure of activities.

4.5.1 Congruence

$(S1)$ $P \equiv P$

$(S2)$ $P \equiv Q \Rightarrow Q \equiv P$

$(S3)$ $P \equiv Q \,,\, Q \equiv R \Rightarrow P \equiv R$

$(S4)$ $P \equiv Q \Rightarrow (P \parallel R) \equiv (Q \parallel R)$

$(S5)$ $P \equiv Q \Rightarrow \alpha \langle \tilde{x} \rangle : \{P\} \equiv \alpha \langle \tilde{x} \rangle : \{Q\}$

4.5.2 Declaration

Local variables and channel names can be introduced and eliminated.

$$(Decl\ -1) \quad \texttt{var}\ \tilde{x}\ \texttt{in}\ \{P\} \equiv P\ (if\ x\ is\ not\ in\ P)$$

$$(Decl\ -2) \quad \texttt{chan}\ \tilde{x}\ \texttt{in}\ \{P\} \equiv P\ (if\ x\ is\ not\ in\ P)$$

$$(Decl\ -3) \quad \texttt{var}\ \tilde{x}\ \texttt{in}\ \{\texttt{var}\ \tilde{y}\ \texttt{in}\ P\} \equiv \texttt{var}\ \tilde{y}\ \texttt{in}\ \{\texttt{var}\ \tilde{x}\ \texttt{in}\ P\}$$
$$\equiv \texttt{var}\ \tilde{x},\ \tilde{y}\ \texttt{in}\ \{\ P\}$$

4.5.3 `delay`

If t is a period of time, then $(\texttt{delay}(t)\ ;\ P)$ is an activity that behaves like P only after the elapse of t time units. If $t\ =\ 0$, it immediately behaves like P.

$$(\texttt{delay}\ -1) \quad \texttt{delay}\ (t_1)\ ;\ \texttt{delay}\ (t_2) \equiv \texttt{delay}\ (t_1\ +\ t_2)$$

$$(\texttt{delay}\ -2) \quad \texttt{delay}\ (0)\ ;\ P \equiv P$$

4.5.4 Deadline

For a given activity P and a set of *duration* S, $[S]\ P$ is an activity in which P has a duration/ deadline $d\ \in\ S$, within which the activity executes and terminates.

An activity with a zero duration is the `skip` activity and that durations can be combined in a sequential composition of activities.

$$(Deadline\ -1) \qquad [0]\ P \quad \equiv \quad \texttt{skip}$$

$$(Deadline\ -2) \qquad [t_P]\ P\ ;\ [t_Q]\ Q \quad \equiv \quad [t_P,\ t_Q]\ (P\ ;\ Q)$$

The deadline of concurrent activities is at most the maximum of the durations.

$$(Deadline\ -3) \qquad [t_P]\ P \parallel [t_Q]\ Q \quad \sqsubseteq \quad [\max\ (t_P,\ t_Q)](P \parallel Q)$$

$$(Deadline\ -4) \qquad [t]\ P \sqsubseteq [t']\ P\ if\ \ t' \subseteq t$$

4.5.5 Sequential

If P and Q are activities with the same alphabet, their composition $(P;\ Q)$ represents a program which runs P first. If P does not terminate, neither does $(P;\ Q)$. Q is started if and when P terminates; and then $(P\ ;\ Q)$ terminates when Q does.

; is associative with `skip` is its unity.

$$(;\ -1) \qquad P\ ;\ (Q\ ;\ R) \quad \equiv \quad (P\ ;\ Q)\ ;\ R$$

$$(;\ -2) \qquad Q\ ;\ \texttt{skip} \quad \equiv \quad Q$$

$$\equiv \quad \texttt{skip}\ ;\ Q$$

4.5.6 Alternative

If P and Q are activities, with the same alphabet, guarded by G_1 and G_2 respectively, the notation $G_1 \rightarrow P \,\Box\, G_2 \rightarrow Q$ describes a program that may behaves as P or as Q, but does not determine which it shall be if both guards are evaluated to `true`, i.e. passable.

When two alternatives are the same activity, the choice becomes vacuous.

$$(Alt-1) \qquad (G \rightarrow P \,\Box\, G \rightarrow P) \;\equiv\; G \rightarrow P$$

$$(Alt-2) \qquad (G \rightarrow P \,\Box\, \texttt{not}\; G \rightarrow P) \;\equiv\; P$$

$$(Alt-3) \qquad (G \rightarrow P \,\Box\, \texttt{not}\; G \rightarrow Q) \;\equiv\; \texttt{not}\; G \rightarrow Q \;\Box\; G \rightarrow P$$

$$(Alt-4) \qquad (c \rightarrow (b \rightarrow P \,\Box\, \texttt{not}\; b \rightarrow Q) \,\Box\, \texttt{not}\; c \rightarrow R) \;\equiv$$

$$(b\;\texttt{and}\;c) \rightarrow P \,\Box\, \texttt{not}\;(b\;\texttt{and}\;c) \rightarrow (c \rightarrow Q \,\Box\, \texttt{not}\; c \rightarrow R)$$

$$(Alt\ -5) \qquad b\ \rightarrow\ P\ \Box\ \texttt{not}\ b\ \rightarrow\ (c\ \rightarrow\ P\ \Box\ \texttt{not}\ c\ \rightarrow\ Q)\ \equiv$$

$$b\ \texttt{or}\ c\ \rightarrow\ P\ \Box\ \texttt{not}\ (b\ \texttt{or}\ c)\ \rightarrow\ Q$$

Proof:

$$L.H.S\ = \qquad (Alt-3)$$

$$\texttt{not}\ b\ \rightarrow\ (\texttt{not}\ c\ \rightarrow\ Q\ \Box\ c\ \rightarrow\ P)\ \Box\ b\ \rightarrow\ P$$

$$= \qquad (Alt-2\ \textit{and}\ Alt-4)$$

$$(\texttt{not}\ c\ \texttt{and}\ b \rightarrow\ Q)\ \Box\ (c\ \texttt{or}\ b\ \rightarrow\ P)$$

$$= R.H.S. \qquad (Alt-3)$$ ■

$$(Alt\ -6) \qquad b \ \rightarrow\ P \,\square\, \texttt{not}\ b \ \rightarrow\ (b \ \rightarrow\ Q \,\square\, \texttt{not}\ b \ \rightarrow\ R) \ \equiv$$

$$b \ \rightarrow\ P \,\square\, \texttt{not}\ b \ \rightarrow\ R$$

Proof:

$L.H.S\ =$ $(Alt-3)$

$b \ \rightarrow\ (\texttt{not}\ b \ \rightarrow\ R \,\square\, b \ \rightarrow\ Q) \,\square\, \texttt{not}\ b \ \rightarrow\ P$

$=$ $(Alt-4)$

$\texttt{false} \ \rightarrow\ Q \,\square\, \texttt{not}\ \texttt{false} \ \rightarrow\ (\texttt{not}\ b \ \rightarrow\ R \,\square\, \texttt{not}\ \texttt{not}(b) \ \rightarrow\ P)$

$=$

$\texttt{false} \ \rightarrow\ Q \,\square\, \texttt{true} \ \rightarrow\ (\texttt{not}\ b \ \rightarrow\ R \,\square\, b \ \rightarrow\ P)$

$=$

$(\texttt{not}\ b \ \rightarrow\ R \,\square\, b \ \rightarrow\ P)$

$=$

$(b \ \rightarrow\ P \,\square\, \texttt{not}\ b \ \rightarrow\ R)$

$=\ R.H.S.$

■

The order in which a nondeterministic choice is made is immaterial.

$$(Alt\ -7) \qquad G_1\ \rightarrow\ P \,\square\, G_2\ \rightarrow\ Q\ \ \equiv\ \ G_2\ \rightarrow\ Q \,\square\, G_1\ \rightarrow\ P$$

Choice can be distributed with the ;

$$(Alt\ -8) \qquad (G_1 \rightarrow R \,\square\, G_2 \rightarrow S)\ ;\ P\ \ \equiv\ \ G_1 \rightarrow (R\ ;\ P) \,\square\, G_2 \rightarrow (S\ ;\ P)$$

We only choose the true guarded activity.

$$(Alt\ -9) \qquad \texttt{true} \rightarrow P \,\square\, \texttt{false} \rightarrow Q\ \ \equiv\ \ P$$

$$(Alt\ -10) \qquad \texttt{false} \rightarrow P \,\square\, \texttt{true} \rightarrow Q\ \ \equiv\ \ Q$$

$$(Alt\ -11) \qquad (G_1\ \rightarrow\ P_1 \,\square\, G_2\ \rightarrow\ P_2)\ \ \sqsubseteq\ \ P_2\ (if\ \ G_2\ \equiv\ \texttt{true})$$

This can be strengthened to: $(G_1 \rightarrow P_1 \,\square\, G_2 \rightarrow P_2)$ is refined by $Q\ if\ G_1 \Longrightarrow G_2$

The nondeterministic choice is disjunctive and associative.

$$(Alt\ -12) \qquad G_1 \rightarrow R \,\square\, G_2 \rightarrow S\ \ \equiv\ \ (G_1\ \texttt{and}\ R)\texttt{or}(G_2\texttt{and}\ S)$$

$$(Alt\ -13) \qquad (G_1\ \rightarrow R \,\square\, G_2\ \rightarrow S) \,\square\, G_3\ \rightarrow\ P\ \ \equiv\ \ (G_1\ \rightarrow R) \,\square\, (G_2\ \rightarrow S \,\square\, G_3\ \rightarrow\ P)$$

$$(Alt\ -14) \qquad (G\ \texttt{and}\ (G_1\ \rightarrow\ P \,\square\, G_2\ \rightarrow\ Q)\ \ \equiv\ \ G\ \texttt{and}\ G_1\ \rightarrow\ P \,\square\, G\ \texttt{and}\ G_2\ \rightarrow\ Q)$$

$$(Alt-15) \qquad a \rightarrow P \,\Box\, (b \rightarrow Q \,\Box\, c \rightarrow R) \;\equiv\; (a \rightarrow P \,\Box\, b \rightarrow Q) \,\Box\, (a \rightarrow P \,\Box\, c \rightarrow R)$$

Proof:

$$\begin{aligned} & R.H.S = && (Alt-2 \textit{ and } Alt-3) \\ & (a \rightarrow P \,\Box\, a \rightarrow P) \,\Box\, (b \rightarrow Q \,\Box\, c \rightarrow R) && \\ & = L.H.S. && (Alt-13) \end{aligned}$$

■

4.5.7 Interrupt

The activity $P \triangleright_t^G Q$ behaves like the activity P for a duration of t time and while G is false. If G remains false for this duration, Q never executes. However, during this duration, if G becomes passable, then it stops behaving like P and instead, it behaves like Q (note that the initial state is restored; i.e. Q will start from the initial state of the original activity, $P \triangleright_t^G Q$. In such a case the duration t is immaterial. Two observations are in order: (a) the computation performed by P before the interrupt are lost and the Q can resume (e.g. to repair the damage done by P), and (b)

This activity distribute through $\|$

$$(Interrupt\ -1) \quad (P \triangleright_t^G Q) \parallel (S \triangleright_t^G T) \equiv (P \parallel S) \triangleright_t^G (Q \parallel T)$$

$$(Interrupt\ -2) \quad P \triangleright_{t+1}^G Q \equiv (P \triangleright_t^G Q) \triangleright_1^G Q$$

$$(Interrupt\ -3) \quad P \triangleright_t^G (Q \triangleright_0^G R) \equiv P \triangleright_t^G Q \ \ (if\ G\ has\ become\ false\ during\ t)$$

$$(Interrupt\ -4) \quad (P \triangleright_t^{\texttt{true}} Q)\ ;\ R \equiv Q\ ;\ R$$

4.5.8 Parallel

$P \parallel Q$ is an activity whose components P and Q are activities running concurrently, with the possibility of communication between them. The activity terminates only when its components terminates.

$$(\parallel -1) \quad P \parallel Q \equiv Q \parallel P$$

$$(\parallel -2) \quad P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R$$

$$(\parallel -3) \quad P \parallel \texttt{skip} \equiv P$$

$$\equiv \texttt{skip} \parallel P$$

An empty $\parallel$ terminates immediately.

$$(\|-4) \quad \langle\,\rangle \,\|\, \langle\,\rangle \equiv \texttt{skip}$$

$$(\|-5) \quad (Chan!v)\ ;\ P \,\|\, (Chan?x)\ ;\ Q \equiv P \,\|x := v\ ;\ Q$$

$$(\|-6) \quad P \,\|\, (G_1 \to R \,\Box\, G_2 \to S) \equiv G_1 \to (R \,\|\, P) \,\Box\, G_2 \to (S \,\|\, P)$$

4.5.9 Assignment

$\tilde{v} := \tilde{e}$ is an activity that assigns the value of an expression list $\tilde{e}$ to the variable list $\tilde{v}$ and then terminate successfully.

The empty multiple assignment terminates with all variables unchanged. (we use $\langle\rangle$ to denote empty list): $(asgn-1) \quad \langle\rangle := \langle\rangle \equiv \texttt{skip}$

The assignment of a variable's own value to itself has no effect. (We use the normal multiple assignments notation, i.e. $(x,y := 1,2)$ means that x and y are assigned to the values 1 and 2 respectively). If $\tilde{x}$ and $\tilde{y}$ are disjoint, then

$$(asgn-2) \quad (\tilde{x},\ \tilde{y} := \tilde{e},\ \tilde{y}) \equiv (\tilde{x} := \tilde{e}).$$

List permutation of variables and expressions are allowed and has no effect the meaning of assignment:

$$(asgn-3) \quad (x,y,z := a,b,c) \equiv (y,x,z := b,a,c).$$

4.5.10 Somewhere

$\texttt{somewhere}(\alpha) \cdot G\,\{P\}$ is an activity which behaves like P only when, somewhere within the context α, the guard G evaluate to `true`.

$$(somewhere\ -1) \quad \texttt{somewhere}(\alpha) \cdot G\,\{P\} \parallel \texttt{somewhere}(\alpha) \cdot G\,\{Q\} \equiv \texttt{somewhere}(\alpha) \cdot G\,\{P \parallel Q\}$$

$$(somewhere\ -2) \quad \texttt{somewhere}(\alpha) \cdot G_1\,\{P\} \parallel \texttt{somewhere}(\alpha) \cdot G_2\,\{P\} \equiv \texttt{somewhere}(\alpha) \cdot (G_1\ \texttt{and}\ G_2)\,\{P\}$$

(Note that we assume that both G_1 and G_2 are true in the same context. However, if both are true but in different places then we have to assert that there will be no place where G_1 `and` G_2 is `true`.)

$$(somewhere\ -3) \quad \texttt{somewhere}(\alpha) \cdot \texttt{true}\,\{\texttt{skip}\} \equiv \texttt{skip}$$

$$(somewhere\ -4) \quad \texttt{somewhere}(\alpha) \cdot \texttt{true}\,\{\texttt{abort}\} \equiv \texttt{abort}$$

(Note that $(Somewhere3\& - 4)$ can be generalised to:

$$\texttt{somewhere}(\alpha) \cdot \texttt{true}\,\{P\} \equiv P)$$

4.5.11 `abort`

`abort` is what we might call a *broken* activity. It will never interact with the outside world and, worst still, the outside world can never detect this fact. Whilst it may be performing some internal actions, the observer cannot disregard that it might still do something!

`abort` is an activity whose behaviour is completely undefined. No other activity share the same characteristics, as it performs internal actions forever attempting to decide what its behaviour will be, but never makes any progress. Worst still, an observer can not see that!

$$(\texttt{abort}-1) \quad \texttt{abort} \equiv \texttt{while true do skip od}$$

$$(\texttt{abort}-2) \quad \texttt{abort} \ ; \ P \equiv \texttt{abort}$$

$$(\texttt{abort}-3) \quad \texttt{abort} \parallel P \equiv \texttt{abort}$$

4.5.12 Loop

$$(Loop-1) \quad P \equiv Q \Rightarrow \texttt{while}\, G\ \{P\} \equiv \texttt{while}\, G\ \{Q\}$$

$$(Loop-2) \quad \texttt{while}\, G\ \{P\} \equiv G \rightarrow P \ ; \ \texttt{while}\, G\ \{P\}$$

4.5.13 Mobility

Uncovering algebraic algebraic characterisation for *mobility*, expressed by the operator "`to` α" (where α is a context), is interesting. For example, we may expect that

$$\beta\,\langle\rangle : \{\texttt{to}\,(\alpha)\ ;\ \texttt{skip}\} \quad \equiv \quad \texttt{skip}$$ which is clearly incorrect.

The algebraic characterisation of mobility has to be operational in nature and will be well understood in a *reduction*-style semantics. In such a semantics, we can formulate a statement such as " when the above activity is released, then performing the step: `to` (α), will lead to it moving to the (new) context, α, where it may be ready to take another step in which it will execute the sub activity `skip`".

However the following is true:

$$\beta\,\langle\rangle : \{\texttt{to}\,(\alpha)\ ;\ \texttt{skip}\} \parallel \alpha\,\langle\rangle : \{\,\} \quad \rightarrow \quad \beta\,\langle\rangle : \{\,\} \parallel \alpha\,\langle\rangle : \{\texttt{skip}\}$$

Similarly, we have

$$\beta\,\langle\rangle : \{\texttt{to}\,(\alpha)\ ;\ \ \texttt{abort} \parallel \alpha\,\langle\rangle : \{\,\} \quad \rightarrow \quad \beta\,\langle\rangle : \{\,\} \parallel \alpha\,\langle\rangle : \{\texttt{abort}\}$$

We note the use of $\rightarrow$ which carries the informal meaning: "reduces to". Similar rules can be constructed as we shall see in Chapter(6) (page 193). However, we can formulate the following rules: "*eliminating brackets*" and "*stuttering*:

$$(Mobility-1) \quad \beta\langle\rangle : \{\texttt{to}\,(\alpha)\,(P\ ;\ Q)\} \equiv \beta\langle\rangle : \{\texttt{to}\,(\alpha)\,P\ ;\ Q\}$$

$$(Mobility-2) \quad \beta\langle\rangle : \{\texttt{to}\,(\alpha)\,(P \parallel Q)\} \equiv \beta\langle\rangle : \{\texttt{to}\,(\alpha)\,P \parallel \texttt{to}\,(\alpha)\,Q\}$$

$$(Mobility-3) \quad \beta\langle\rangle : \{\texttt{to}\,\beta\ P\} \equiv \beta\langle\rangle : \{P\}$$

Another interesting phenomenon with the mobility operator is its "*hopping*" and "*discrete*" behaviours: For example

$$(\texttt{to}\,(\beta)\,(\texttt{to}\,(\alpha)\ P)) \rightarrow \texttt{to}\,(\alpha)\,P$$

Here, the activity moves to β, then to α and then returns back to α. This raises the question of discreetness: If an activity moves from its current context (say α) to another context, β, does it mean, there must have been a number of other (hidden/implicit) contexts where the activity has gone through before reaching its final context? In our model, this is not necessarily true as contexts are uniquely "named" and hence have no "structure" at all. However, hopping may be used to model proximity of contexts and neighborhoods.

4.5.14 Algebraic Structures

The algebraic rules given above provide a useful tool to manipulate "structures" of $\mathcal{CS}-\mathcal{F}low$ models. These laws however induce complete algebraic "*characterisation*" for $\mathcal{CS}-\mathcal{F}low$. These characterisations comes in the form of known structures.

Let $\mathcal{A}$ be the set of activities, then (given $\texttt{zero} \mathrel{\widehat{=}} \texttt{not}\ (\texttt{skip}\ ;\ \texttt{delay}(0))$):

1. $(\mathcal{A},\ \|,\ ;\ ,\ \Box,\ \texttt{zero},\ \texttt{delay}(0),\ \texttt{abort})$ is an idempotent left semiring :
   - $(\mathcal{A}, \|, \texttt{delay}(0))$ is a commutative monoid ($\| -1$, $\| -2$, and $\| -3$),
   - $(\mathcal{A}, ;, \texttt{zero})$ is a monoid ($; -1$ and $; -2$);
   - $(\mathcal{A}, \Box, \texttt{abort})$ is a commutative monoid.
2. $(\mathcal{A}, \texttt{not})$ is a Boolean idempotent left semiring because of
   - (1) above and
   - $b_1 = \texttt{not}\ ((\texttt{not}(b_1))\ \texttt{or}\ (\texttt{not}\ b_2))\ \texttt{or}\ \texttt{not}((\texttt{not}(b_1))\ \texttt{or}\ b_2)$
   - $(a\ \texttt{and}\ b) = \texttt{not}\ ((\texttt{not}\ a)\ \texttt{or}\ (\texttt{not}\ b))$
   - $\texttt{true} = a\ \texttt{or}\ (\texttt{not}\ a)$
   - $\texttt{false} = a\ \texttt{and}\ (\texttt{not}\ a)$
3. We also have the usual properties relating to Boolean Kleene algebra.

4.5.15 example

The above algebraic laws can be used to structurally transform a given model/specification to an equivalent one whose properties are known. As a simple, but illustrative example, the model

(4.1) $$\texttt{var}\ x\ \texttt{in}\ \{\ x := v\ \}$$

is deadlock-free. We can utilise the above laws to infer that the following model (Eq(4.2) is also deadlock-free:

(4.2) $$\texttt{chan}\ c\ \texttt{in}\ \{\ c\ !\ v \parallel (\texttt{var}\ x\ \texttt{in}\{c\ ?\ x\})\ \}$$

The transformation proceeds as follows

$$\begin{aligned}
Eq(4.2) &\equiv (decl-3)\\
&\quad \texttt{chan}\ c\ \texttt{var}\ x\ \texttt{in}\ \{\ c\ !\ v \parallel c\ ?\ x\ \}\\
&\equiv (\parallel -5)\\
&\quad \texttt{chan}\ c\ \texttt{var}\ x\ \texttt{in}\ \{\texttt{skip} \parallel x := v\ \}\\
&\equiv (\parallel -3)\\
&\quad \texttt{chan}\ c\ \texttt{var}\ x\ \texttt{in}\ \{\ x := v\}\\
&\equiv (decl-1)\\
&\quad \texttt{var}\ x\ \texttt{in}\ \{\ x := v\ \}\\
&\equiv Eq(4.1)
\end{aligned}$$

■

4.6 Timing Behaviours

4.6.1 introduction

As we mentioned earlier, activities in workflows can be time-dependent and it was important in the design of $\mathcal{CS}-\mathcal{F}low$ to make appropriate provisions by providing constructs such as `delay` and [S] P. In this section we give some of the important and useful timing characterisation of activities. Once again, let us recall that we let t represents time, such that $t \in \mathcal{N}^3$ and that t_α and t_β denotes the *start* and *termination* time of an activity, such that

$$t_\alpha \leq t_\beta.$$

(i.e. We also assume that the termination time of an activity will be at, or after its release time.). Let us also use `at` to be a function that returns a value of a state variable at a particular "point" in time: The `at` mapping is defined as

$$\texttt{at}: \quad Var \times \mathcal{N} :\rightarrow \mathcal{N}$$

For example for the activity $z := x + y$,

$$z \ \texttt{at} \ (t_\beta) = x \ \texttt{at} \ (t_\alpha) + y \ \texttt{at} \ (t_\alpha)$$

[3]The time domain is modelled by $(\mathcal{N}, \leq)$, where $\leq$ is a total order

We can describe the "stability" of a variable X using a time predicate $\mathcal{P}(t)$ which has a single free time variable:

$$\mathcal{P}(t) \mathrel{\widehat{=}} X \ \texttt{at} \ (t) \ = \ X \ \texttt{at} \ (t-1)$$

Various important timing properties can be defined. Here we consider *duration* ($\mathcal{D}$), *within* ($\mathcal{W}$), *after* ($\mathcal{A}$), *between* ($\mathcal{B}$) and *every* ($\mathcal{E}$).

Definition 4.6.1.1

$$\mathcal{D} \ \odot \ n \tag{4.3}$$

This is defined for any binary relation, $\odot$, on positive integers and asserts various constraints over the execution interval.

$$\mathcal{W}\,(n, t_\alpha, t_\beta, \mathcal{P}(t)) \mathrel{\widehat{=}} \bigvee_{i \in [t_\alpha, \min(t_\alpha + n,\, t_\beta)]} \mathcal{P}[i/t] \tag{4.4}$$

$\mathcal{W}$ asserts that $\mathcal{P}(t)$ is valid at a point in time within the first n time unites after the activity was released.

$$\mathcal{A}\,(n, t_\alpha, t_\beta, \mathcal{P}(t)) \mathrel{\widehat{=}} \bigvee_{i \in [t_\alpha + n,\, t_\beta]} \mathcal{P}[i/t] \tag{4.5}$$

$\mathcal{A}$ asserts that $\mathcal{P}(t)$ holds true at a point in time after the first n time unites from the activity's release.

$$\mathcal{B}\ (\mathcal{P}(t), t_\alpha, t_\beta, n, Q(t)) \ \widehat{=} \bigwedge_{i \in [t_\alpha,\, t_\beta - n]} \mathcal{P}[i/t] \ \Rightarrow \bigvee_{j \in [i\ \min(i+n,\, t_\beta)]} Q[j/t] \tag{4.6}$$

This asserts that for every time for which $\mathcal{P}(t)$ is true there must be a time, not more than n time unites away, at which $Q(t)$ is true.

$$\mathcal{E}\ (n, t_\alpha, t_\beta,\ \mathcal{P}(t)) \ \widehat{=} \bigwedge_{i \in Pr(t_\alpha, t_\beta, n)} \mathcal{W}\ (n, \mathcal{P}(t))[i/t_\alpha, i+n/t_\beta] \tag{4.7}$$

where

$$Pr(a,b,n) \ \widehat{=} \ \{i\ :\ i \in [a,b] \wedge \exists\, l\ ((n.l) + a = i) \wedge n + i \leq b\}$$

This asserts that for at least one instant of time within each consecutive time interval of length n, $\mathcal{P}(t)$ must be true (except for the remainder of the interval if it is less than n time unites). ■

4.6.2 Properties

We can derive a number of proof rules that governs these timing predicates. Our rational for this is to be able to formally prove some interesting timing properties for activities.

We adopt the general form of a rule:

$$\frac{\begin{array}{l} C_1, \\ C_2, \\ (Name)\ \ldots \\ C_n \end{array}}{Design/model \models prop}$$

where C_i is a constraints. The rule states that given the constraints, then the $Design/model$ satisfies the $prop$.

4.6.3 duration – $\mathcal{D}$

Let $primitive \in \{!, ?, x := v, \}$. We have

$$(Dur-0)\ \frac{}{primitive \models \mathcal{D} \geq 1}$$

Any primitive construct should take one or more 1 time unite.

$$(Dur-1)\ \frac{\begin{array}{l} P \models \mathcal{D} \geq n, \\ Q \models \mathcal{D} \geq m \end{array}}{P\ ;\ Q \models \mathcal{D} \geq (n+m)}$$

The sequential composition of two activities takes at least the sum of the durations of both.

$$(Dur-2)\ \frac{P \models \mathcal{D} \geq n,\quad Q \models \mathcal{D} \geq m}{P \parallel Q \models \mathcal{D} \geq \max(n,m)}$$

The parallel composition of two activities takes at least the largest duration of both.

$$(Dur-3)\ \frac{P \models \mathcal{D} \geq n}{[S]\,P \models \mathcal{D} \geq n}$$

The duration of an activity (if known) is at least one of its allocated (at design phase) deadlines.

$$(Dur-3a)\ \frac{P \models \Phi}{[S]\,P \models \Phi}$$

$$(Dur-4)\ \frac{}{(Dur-4)\ [S]\,P \models \mathcal{D} \in S}$$

However, if its duration is not known, then it must be one of those that was allocated at design time (i.e. one in S).

$$(Dur-5)\ \frac{P \models \mathcal{D} \geq n,\quad G}{(\texttt{while}\ G\ \texttt{do}\ P\ \texttt{od}) \models \mathcal{D} \geq n}$$

For as long as we know the duration of an activity, we can infer the duration of its repetitive behaviour.

$$(Dur-6)\ \frac{\begin{array}{l} P \models \mathcal{D} \geq n, \\ Q \models \mathcal{D} \geq m, \\ G_1 \texttt{ at } t_\alpha \vee G_2 \texttt{ at } t_\alpha \end{array}}{(G_1 \rightarrow P \,\Box\, G_2 \rightarrow Q) \models \mathcal{D} \geq \min(n,m)}$$

The duration of the conditionals is at least the duration of its minimum; and it is obviously true that

$$(Dur-7)\ \frac{Q \models \mathcal{D} \geq n}{(P \rhd_t^{\texttt{false}} Q) \models \mathcal{D} \geq (n+t)}$$

$$(Dur-8)\ \frac{}{\texttt{delay}(n) \models \mathcal{D} \geq n}$$

4.6.4 within – $\mathcal{W}$

$$(within-1)\ \frac{P \models \mathcal{W}(n, \mathcal{P}(t))}{P \,;\, Q \models \mathcal{W}(n, \mathcal{P}(t))}$$

$$(within-2)\ \frac{P \models \mathcal{W}(n, \mathcal{P}(t))}{[S]P \models \mathcal{W}(n, \mathcal{P}(t))}$$

$$(within-3)\ \frac{P \models \mathcal{W}(n, \mathcal{P}(t)),\quad P \models \mathcal{D} = (n+m)}{P \models \mathcal{W}(n+m,\ \mathcal{P}(t))}$$

$$(within-3a)\ \frac{P \models \bigvee_{t \in [t_\alpha, t_\beta]} \mathcal{P}(t)\quad P \models \mathcal{D} \leq n}{P \models \mathcal{W}(n, \mathcal{P}(t))}$$

$$(within-4)\ \frac{P \models \mathcal{D} \leq n,\quad P \models \bigvee_{\sigma \in [t_\alpha, t_\beta]} \mathcal{P}(t)}{P \models \mathcal{W}(n, \mathcal{P}(t))}$$

$$(within-5)\ \frac{P \models \mathcal{D} \leq n}{P \,\|Q \models \mathcal{W}(n, \mathcal{P}(t))}$$

4.6.5 after – $\mathcal{A}$

$$(after-1)\frac{P \models \mathcal{A}(n, \mathcal{P}(t))}{P\ ;\ Q \models \mathcal{A}(n, \mathcal{P}(t))}$$

$$(after-2)\frac{P \models \mathcal{A}(n, \mathcal{P}(t))}{P \,\|Q \models \mathcal{A}(n, \mathcal{P}(t))}$$

$$(after-3)\quad \frac{Q \models \mathcal{A}(n,\mathcal{P}(t)),\quad P \models \mathcal{D} \geq m}{P\ ;\ Q \models \mathcal{A}(n+m,\mathcal{P}(t))}$$

4.6.6 between – $\mathcal{B}$

$$(between-1)\quad \frac{A \models \mathcal{B}(P(t),n,Q(t))}{A \parallel B \models \mathcal{B}(P(t),n,Q(t))}$$

$$(between-2)\quad \frac{A \models \mathcal{B}(P(t),n,Q(t))}{A\ ;\ B \models \mathcal{B}(P(t),n,Q(t))}$$

$$(between-3)\quad \frac{A \models \mathcal{B}(P(t),n,Q(t)),\quad B \models \mathcal{B}(P(t),m,Q(t)),\quad m \leq n}{A\ ;\ B \models \mathcal{B}(P(t),n,Q(t))}$$

$$(between-4)\quad \frac{A \models \mathcal{B}(P(t),l,Q(t)),\quad B \models \mathcal{B}(P(t),l,Q(t)),\quad A \models \mathcal{D} = n,\quad B \models \mathcal{D} = n}{A \parallel B \models \mathcal{B}(P(t),l,Q(t))}$$

4.6.7 every – $\mathcal{E}$

$$(every-1)\quad \frac{\begin{array}{l} m \geq n, \\ P \models \mathcal{A}(n, \mathcal{P}(t)), \\ P \models \mathcal{D} = m \end{array}}{(\texttt{while } G \texttt{ do } P \texttt{ od}) \models \mathcal{E}(m, \mathcal{A}(n, \mathcal{P}(t)))}$$

$$(every-2)\quad \frac{\begin{array}{l} P \models \mathcal{E}(n, \mathcal{P}(t)), \\ Q \models \mathcal{E}(m, Q(t)) \end{array}}{(P \parallel Q) \models \mathcal{E}(n, \mathcal{P}(t)) \wedge \mathcal{E}(m, Q(t))}$$

$$(every-3)\quad \frac{\begin{array}{l} P \models \mathcal{D} = l, \\ \exists m \bullet (m.n = l), \\ P \models \mathcal{E}(n, \mathcal{P}(t)), \\ Q \models \mathcal{E}(n, \mathcal{P}(t)) \end{array}}{(P \,;\, Q) \models \mathcal{E}(n, \mathcal{P}(t))}$$

$$(every-4)\quad \frac{\begin{array}{l} P \models \mathcal{W}(l, \mathcal{P}(t)), \\ P \models \mathcal{D} = m \end{array}}{([m.n]\ \texttt{while } G = n \texttt{ do } P \texttt{ od}) \models \mathcal{E}(m, \mathcal{W}(l, \mathcal{P}(t)))}$$

4.6.8 General Rules

In addition to the above, there are some (traditional) general rules which are useful.

$$\frac{P \models \Phi \qquad (Weaken)\ \Phi \Rightarrow \Psi}{\Psi \models P}$$

$$\frac{P \models \Phi \qquad (and)\ P \models \Psi}{P \models \Phi \wedge \Psi}$$

4.7 Summary

In the previous chapter we have motivated the need for a novel design notation for workflow systems. Our arguments stem from the realisations that current workflow systems are highly distributed, cross boundaries of variety of organisation/enterprises and above all have timing characteristics.

Therefore any new notation/language has to enjoy the following features:

- support concurrency;
- context and context awareness are first-class citizen;
- supports mobility as activities can move from one context to another;
- has the ability to express timing constrains: delay, deadlines, priority and schedulability;

- allows the expressibility of (*access control*) security policies without the need for an extra linguistic complexities;

- enjoy sound formal semantics that allows us to animate design and compare various designs.

In this chapter we have given, a formal semantics to $\mathcal{CS}-\mathcal{F}low$, which satisfies all of the above. We also gave an algebraic characterisation to the language which will give us the mechanism for analyses and proofs.

Chapter 5

$\mathcal{CS}-\mathcal{F}low$

CLOSED LAYERS: Design Principle

Objectives

- Discuss the rational for the design principle of closed layers and give a procedure for constructing them.
- Provide a development life cycle for layer design.
- Explore the design of fault-tolerant workflows using Closed layers.

5.1 Introduction

Designing and analysing software system is intrinsically hard. In fact the quest for an engineering process has been of concern to theoreticians and practitioners alike for many decades. And still is an active area of research and development. From, what has been termed "traditional" development process, the so called *waterfall*, to *agile* development process, through, *rapid prototyping* and *extreme programming and development*.

The reason for this is that a development process should deal with not only the mere construction of the artifacts but also our ability to analysis their current behaviours and predicting their future ones.

Currently, practitioners within software houses and industries, in general (and unlike those in more traditional engineering disciplines), adopt what they see fit in the development given various constraints such as time, budget, available expertise, etc. Even the sharing of best practices amongst them is hardly practiced.

This is certainly true for transformational (closed) systems in which their behaviours can be fully determined by the relationship between their input and output – for example, compilers, payroll systems, etc. The situation becomes much harder when developing concurrent/distributed systems, and even more so when considering those of reactive characteristics such as what is now commonly known as ubiquoutus systems – systems which are context-aware.

As we have already established, workflow systems are typical example of this type and they are even harder as the majority of them have timing characteristics as well as being highly distributed. For example, some workflows within a typical hospital ward may have to satisfy some hard real-time constraints, such as insulin must be injected at some regular intervals.

In addition, being a transactional systems, some workflow systems can be of long-running transaction in which dependability requirements are crucial. These requirements vary from being highly reliable and available, to being safe and secure. For example, a context-aware system for an operating room in a hospital, must have a general awareness of the working context, such as staff, patient, equipment, and medical material. From such context information, the system should be able to provide the surgical team with important clinical data correctly and at the appropriate moment (such as medical images and medical records), as well as to detect potential safety critical situations, such as the wrong patient is detected or if the surgery is started before the team is ready.

Given the complex nature of current workflows, being highly distributed, secure-critical and being context-aware, the quest for a design technique that help to increase its reliability becomes crucial. This chapter describes a design technique that will allow the design of a reliable, context and secure workflow. The approach is based on the notion of communication-closed layers. The work on communication-closed layer was first presented by Elrad and Francez [45] as a

notion for analysing distributed systems with message passing as its communication mechanism. Later, it was adapted for scheduling real-time tasks in a tightly-bound environment [44]. Various authors, [46, 77, 107, 150, 151], have studied the notion from a theoretical view point. In this chapter, we review the notion of **closed** layer in general and present it as

- a general design principle of $\mathcal{CS}-\mathcal{F}low$ workflow applications;
- a base for a high-level programming construct which can offer a linguistic support for the design of reliable (fault-tolerant) workflow systems; and
- an efficient approach for the analyses of workflows.

We begin in Section(5.2) with an explanation what a layer is and how it can be systematically constructed. We distinguished between communicating and non-communicating layers and that under a certain condition, a communicating layer can be communication-closed. Then we give a constructive technique by which a semantically-equivalent communication-closed layer system (that is quasi- sequential) to the original system. We have also elaborated on the fact that in the presence of timing constrains, time-closeness is also required and give a process to construct time-closed layers. We end the chapter with a disscusion, Section(5.5) on how layers can be be utilised efficiently to design a reliable (fault-tolerant) $\mathcal{CS}-\mathcal{F}low$ workflow systems.

5.2 Layer Construction

Let us consider the following general workflow in $\mathcal{CS}-\mathcal{F}low$, Table (5.1), which has two concurrent activities $\mathcal{P}$ and $\mathcal{Q}$:

$$C_1\ \langle \tilde{a} \rangle : \{\ \mathcal{P}\ \} \parallel C_2\ \langle \tilde{b} \rangle : \{\ \mathcal{Q}\ \}$$

such that

- C_1 and C_2 are context identifications and that $\tilde{a}$ and $\tilde{b}$ are their sets of attributes.

- $\mathcal{P}$ and $\mathcal{Q}$ are activities.

Specifically, we have[1]

[1] We opted for a *vertical* representation of workflow only for clarity. *Horizontal*-type of representation may be economical but might not be as clear.

Table 5.1 – General Structure in $\mathcal{CS}-\mathcal{F}low$

$C_1\ \langle \tilde{a} \rangle$:		$C_2\ \langle \tilde{b} \rangle$:
{		{
var $\tilde{x}$, $\tilde{y}$ in		var $\tilde{x_1}$, $\tilde{y_1}$ in
{		{
P_1;		Q_1;
P_2;		Q_2;
P_3;	$\parallel$	Q_3;
P_4;		
P_5;		
}		}
}		}

Assumptions. In order to simplify the presentation, and without lose of generality, we assume that

- There is only one parallel operator, $\parallel$, in our system. Nesting concurrency can be dealt with by applying the transformation to the most inner $\parallel$ and continue to move to the outer constructs. For example, if $Q_3 \ \widehat{=}\ Q_{31} \parallel Q_{32}$, then we apply our layering technique to Q_3 and then move upwards.

- The length of all activities in the system are the same. I.e. each of the activities under consideration has the same number of sub activities. This

can be easily achieved using the semantics of `skip`. I.e.

$$\texttt{skip}\ ;\ S \equiv S\ ;\ \texttt{skip} \equiv S$$

The system above is equivalent to the one in Table 5.2.

Table 5.2 – General Structure in $\mathcal{CS}-\mathcal{F}low$

$C_1\ \langle \tilde{a} \rangle$:		$C_2\ \langle \tilde{b} \rangle$:
{		{
var $\tilde{x}$, $\tilde{y}$ in		var $\tilde{x_1}$, $\tilde{y_1}$ in
{		{
P_1;		skip;
P_2;		Q_1;
P_3;	$\parallel$	skip;
P_4;		Q_2;
P_5;		Q_3;
}		}
}		}

In another words, we can pad the shorter with as many `skip`. We present a transformation process that transform a given concurrent system into what we call *quasi-parallel* system. This provides a **super-structure** over the usual process-based structure. Such a super-structure will both facilitated the design and the analysis of workflow systems and also provides a natural way of achieving fault-tolerance.

As a simple but illustrative running example, let us consider the following running $\mathcal{CS}-\mathcal{F}low$ workflow which has two concurrent activities $\mathcal{P}$ and Q:

$$\mathcal{R} \;\widehat{=}\; C_1\ \langle \tilde{a} \rangle : \{\ \mathcal{P}\ \} \parallel C_1\ \langle \tilde{b} \rangle : \{\ Q\ \}$$

P and Q are given in Table(5.3)

Table 5.3 – The $\mathcal{P}$ and Q in $\mathcal{R}$

$C_1\ \langle \tilde{a} \rangle :$		$C_2\ \langle \tilde{b} \rangle :$
{		{
var x, y, z, $chan_1$ in		var x_1, $chan_1$ in
{		{
$y := y + x;$		$chan_1 \,?\, x_1;$
$z := y \times z;$	$\parallel$	$x_1 := x_1 \times x_1;$
$chan_1 \,!\, z;$		
}		}
}		}

To begin with, let us define few important terms:

Definition 5.2.0.1 *A layer, L of a workflow, $\mathcal{S}$, is a logical horizontal partition that cut across all concurrent threads of $\mathcal{S}$* ■

.

Definition 5.2.0.2 *A super-structure over a workflow, $\mathcal{S}$, is a quasi-sequential composition of layers from $\mathcal{S}$* ■

.

The *super-structures* are in fact "safe" decomposition of our original workflow.

Let us now design a number of super-structures over our workflow $\mathcal{R}$. We should note that the "design" is by construction. Our construction starts with extracting a number of layers from which we build the super-structure(s). (**Note** that we have deliberately removed all declarations to simplify the presentation.)

Let us consider the following layers:

Table 5.4 – Some Layers

$L_1 \mathrel{\widehat{=}}$	$y := y + x$	$\|\|$	$chan_1 ? x_1$
$L_2 \mathrel{\widehat{=}}$	$z := y \times z$	$\|\|$	$x_1 := x_1 \times x_1$
$L_3 \mathrel{\widehat{=}}$	$chan_1 ! z$	$\|\|$	`skip`
$L_4 \mathrel{\widehat{=}}$	$y := y + x$; $z := y \times z$	$\|\|$	`skip`; `skip`
$L_5 \mathrel{\widehat{=}}$			
	$z := y \times z$; $chan_1 ! z$; `skip`	$\|\|$	`skip` ; $chan_1 ? x_1$; $x_1 := x_1 \times x_1$

Nesting layer can be also constructed as long the closeness property is preserved (to ensure safety de-composition). Let us consider our L_5 in the above workflow, $\mathcal{R}$. We can clearly construct (decompose) another layer:

Table 5.5 – L_6: Communicating Layer

$chan_1 \,!\, z;$	$chan_1 \,?\, x_1;$

and L_5 can now be written as:

Table 5.6 – L_5: Layer Nesting

$z := y \times z;$		`skip;`
$L_{6.1};$	$\parallel$	$L_{6.2};$
`skip;`		$x_1 := x_1 \times x_1$

The following are some super-structures:

1. $\mathcal{S}_{\mathcal{R}_1} \mathrel{\widehat{=}} \mathcal{R}$
2. $\mathcal{S}_{\mathcal{R}_2} \mathrel{\widehat{=}} L_1 \;;\; L_2 \;;\; L_3$
3. $\mathcal{S}_{\mathcal{R}_3} \mathrel{\widehat{=}} L_4 \;;\; L_5$

It is obviously clear that, for an $\mathcal{CS}-\mathcal{F}low$ workflow $\mathcal{S}$, there are $n!$ of super-structures, where n is the sum of the lengths of all its concurrent threads. An important question will therefore be:

Under what condition(s) will a super-structure workflow be equivalent to the original one?.

In our running example, which super-structure, $\mathcal{S}_{\mathcal{R}_2}$ or $\mathcal{S}_{\mathcal{R}_3}$, is equivalent to $\mathcal{R}$?

For this we need the following preliminaries.

Definition 5.2.0.3 *A Layer L is called communicating layer if it contains at least one communication primitive. It is called communication-closed if a communication starts and terminates in the same layer.*

A non-communicating layer is that which contains no communication primitives.

The various layer structures are depicted in Figure(5.1).

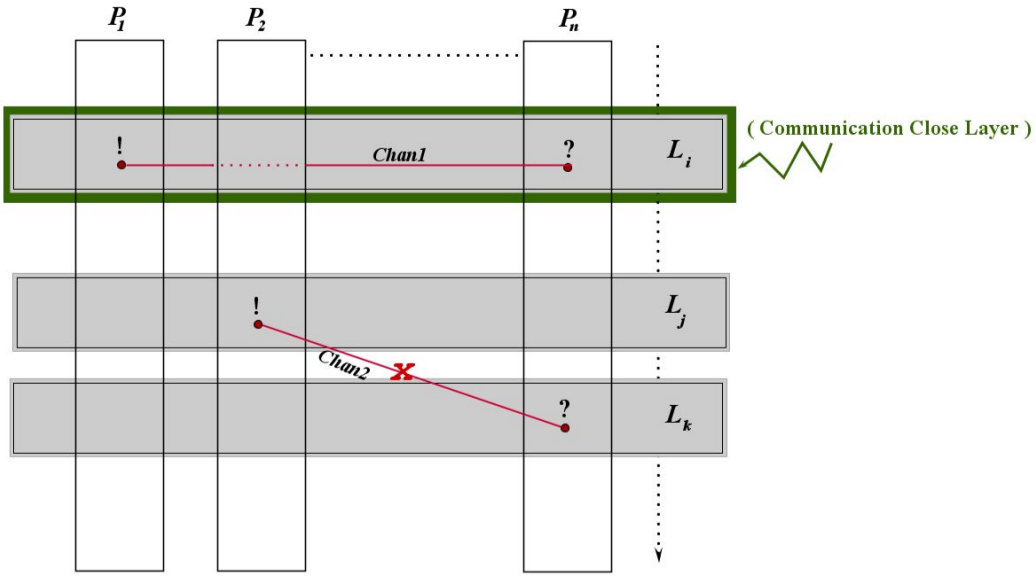


Figure 5.1 – Layers

It is clear that, in the example above, L_2 and L_4 are non-communicating layers while L_1, L_3 and L_5 are communication-closed. $\mathcal{S}_{\mathcal{R}_3}$ and $\mathcal{S}_{\mathcal{R}_1}$ are a quasi-sequential workflow whilst $\mathcal{S}_{\mathcal{R}_2}$ is not.

5.3 Dealing with Choices and Iterations

The decomposition of a given $\mathcal{CS}-\mathcal{F}low$ system, requires a process to deal with structures such as choices and iterations. In this section we study the decomposition of these structures.

5.3.1 Process for Choices

Let us consider the general form of a *choice*. Here, we consider the non-deterministic case as proioritised choices are straightforward. We also omit any declaration to simplify the presentation. We begin by a procedure that *unravels* choices. We should recall that that guards are purely Boolean variables and are not communicating guards (unlike languages such as OCCAM/CSP [62]). Let us consider the following guarded-choice workflow $\mathcal{S}$ in which one or all of its sub activities contain communication activities with other workflow of the same (or different structure) [2]. Let us consider the following workflow, (S),

Table 5.7 – Choices: $\mathcal{S}$

$G_1 \rightarrow P$
$\Box$
$G_2 \rightarrow Q$

[2]The most interesting case is that where we have the same structures

The question here is: *can we transform $\mathcal{S}$ to a semantically equivalent workflow, i.e. which is composed of a communication-closed layer structure?*[3]. The following gives a procedure that enables us to do so.

The following workflow, $\mathcal{S}'$, is a such safe decomposition:

$$\mathcal{S}' \mathrel{\widehat{=}}$$

$$\left\{ \begin{array}{ll} \left[\begin{array}{l} G_1 \rightarrow P \ ; \ GFlag := \texttt{false} \\ \Box \\ G_2 \rightarrow GFlag := \texttt{true} \end{array} \right] \ ; & (S_{11}) \\ \\ \left[\begin{array}{l} GFlag \rightarrow Q \\ \Box \\ \texttt{not}\ GFlag \rightarrow \texttt{skip} \end{array} \right] & (S_{12}) \end{array} \right\}$$

We can see that $\mathcal{S} \equiv \mathcal{S}'$ for the following reason. There are four cases:

- G_1 and G_2 are both false: then the first branch (S_{11}) in the sequence will `abort` as required by the semantics,
- G_1 is `true` and G_2 is `false`, then P is executed and $GFlag$ is set to `false`, and the second branch (S_{12}) executes to `skip`,
- G_1 evaluates to `false` and G_2 to `true`, then P will not be executed, $GFlag$

[3] This is known as **safe** decomposition, [45]

will be then set to `true`, and the second branch executes Q as required,

- G_1 is evaluated to `true` and that G_2 is evaluated to `true` then first branch chooses whether to execute P and the set $GFlag$ to `false`, or set $GFlag$ to `true`. The second branch only executes Q if $GFlag$ is `true` (i.e. only if the first branch choses not to execute P).

Now, if we have another workflow $\mathcal{D}$ of the same structure as $\mathcal{S}$ then

$$\mathcal{S} \parallel \mathcal{D}$$

can be "safely" decomposed into the structure:

$$((\mathcal{S}_{11} \parallel \mathcal{D}_{11}) \; ; \; (\mathcal{S}_{12} \parallel \mathcal{D}_{12})) \; \square \; ((\mathcal{S}_{21} \parallel \mathcal{D}_{21}) \; ; \; (\mathcal{S}_{22} \parallel \mathcal{D}_{22})) \quad (1)$$

where each $\mathcal{S}_{ij}$, for all $i, j = 1, 2$, is either a non-communicating layer or a communication-closed layer.

Equation (1) can be rewritten as

$$((\mathcal{S}_{11} \parallel \mathcal{D}_{11}) \square (\mathcal{S}_{21} \parallel \mathcal{D}_{21}));$$

$$((\mathcal{S}_{11} \parallel \mathcal{D}_{11}) \square (\mathcal{S}_{22} \parallel \mathcal{D}_{22}));$$

$$((\mathcal{S}_{12} \parallel \mathcal{D}_{12}) \square (\mathcal{S}_{21} \parallel \mathcal{D}_{21}));$$

$$((\mathcal{S}_{12} \parallel \mathcal{D}_{12}) \square (\mathcal{S}_{22} \parallel \mathcal{D}_{22})) \quad (2)$$

The structures (1) and (2) demonstrate that layers can be composed, respectively,

as a series of alternative or sequentially. In fact, structures such as iteration, conditional, interrupt, etc. can also be used.

5.3.2 Process for Iterations

We assume that

1. Loops are *finite*

2. Communication symmetry is assured (i.e., communication-deadlock free)

It should be noted that due to (1) above, a finite loop can be replaced as a set of sequentially composed statements and because of (2), we can always ensure (using `skip`) that each layer is communication-closed layer. Therefore, using

$\texttt{while}\ G\ do\{P\}\ \equiv\ G\ \rightarrow\ P\ \ ;\ \ (\texttt{while}\ G\ do\{P\})$

then, if we have

$$\texttt{while}\ G\ do\{P\} \| Q$$

Then we can transform this to the semantically equivalent $\mathcal{CS}-\mathcal{F}low$ system

$$((G\ \rightarrow\ P)\ \|\ Q)\ ;\ (\texttt{while}\ G\ do\{P\})$$

Then, we layer $((G\ \rightarrow\ P)\ \|\ Q)$ into communication-closed layers (depending on the structure of P and Q, and repeat the process on the $\texttt{while}\ G\ do\{P\}$, and so on.

Theorem 5.3.1 *For any $\mathcal{CS}-\mathcal{F}low$ workflow system $\mathcal{S}$ there exist a semantically equivalent quasi-sequential system, $\mathcal{S}_{\mathcal{L}}$.*

Proof: The proof is by construction. Let $\mathcal{S} \mathrel{\widehat{=}} S_1 \,\|S_2\|\, \cdots \,\|\, S_n$. The construction follows three basic steps:

- *Padding*. Ensure that all S_i are of the same length [4], using `skip`, applying rule (; -2) and ($\|$ -3)

- Locate correspondence communication activities and form a communication layer. Others can form non-communicating layers. If communications appear in non-deterministic activity, apply the transformation given in Section(5.3).

- Resolve communication using rule ($\|$ -5). ■

Notes

- As most, if not all, workflow systems have timing constraints, the communication-closed layer notion needs to be modified and introduce what might be called time-closed layer:

  $$[t]\ S$$

  is transformed to

  $$[t_1]\ L_1\ ;\ [t_2]\ L_2\ ;\ \cdots\ [t_n]\ L_n$$

  such that $t\ \geq \sum t_i$; this needs further development.

[4] I.e. each S_i has the same number of sub activities

- Existing proof systems can be utilised on the resulting quasi-sequential system. For example, Haore's triple formalism can be readily applied:

$$\textit{if } \{p_1\}\ L_1\ \{q_1\} \textit{ and } \{q_1\}\ L_2\ \{q_2\} \textit{ then } \{p_1\}\ (L_1\ ;\ L_2)\ \{q_2\}$$

  The development of such proof system as well as high-level linguistic support for layer constructs are some of issues for future work (Chapter(8), page 247).

5.4 Layer Design Methodology

To utilize the concept of *closed-layers*, we need a layer-design methodology.
This section outlines our methodology and gives a simple but illustrative example to demonstrate its use. However, the example is a generic and can be utilise in the design of larger $\mathcal{CS}-\mathcal{F}low$ systems.
To begin with, we have to distinguish between two distinct directions where *Layers* can be used:

- **Analysis.** In this direction, we assume the existence of an $\mathcal{CS}-\mathcal{F}low$ system which is to be analyse. *Analysis* here involves the whole spectrum of validation and verification: from animation/simulation and run-time validation to formal proofs and/or model checking of some known properties (e.g. safety and/or liveness properties [121, 123, 131, 133]; as a simple example

is to demonstrate its absence of deadlocks.

The idea here is to transform the existing $\mathcal{CS}-\mathcal{F}low$ design into a semantically equivalent communication-closed layer design in which the analyses are easier than the original one. The rational is that the resulting layer-design is quasi-sequential and hence all existing formalisms for sequential systems can be deployed. (for example, Hoare's logic, Dijkstra's pre/post condition formalisms, etc.)

- **Design.** Here we assume a general statement of system's requirements, which we can be decomposed into a set of layer requirements and proceed in a traditional fashion. In this case, we apply the following phases:

  1. **Requirements Decomposition.** In this phase, we decompose the given workflow requirements into a number of sub-requirements. These sub-requirements can be as fine or coarse grain as we wish. This is the most difficult phase in the development. This is because, for an efficient decomposition of the requirements, we rely on experience and understanding the none-functional requirements of the system, etc. It is therefore important that such a phase has to be *iterative* in nature.

     Experience has shown that, identifying, what we call *Actors* helps in specifying layer interfaces.

     In addition, this phase also involve the identification of all channels

that may be needed for communications, and also all of the state variables of interest (for these are fundamental to study behaviours).

2. **Layer Design.** In this phase, we design layers which conform/satisfy its requirement. The layers however have to be *communication-closed* layers. For conformity and/or satisfaction, we deploy classical/well established techniques.

3. **Integration.** In this phase we compose/integrate all layers into a complete $\mathcal{CS}-\mathcal{F}low$ workflow.

In the remainder of this section, we illustrate the layering development on one of our previous example given in Section(3.5).

5.4.1 Example

Here we consider the one-place buffer.

1. Requirements Decomposition. In this example, we can easily identify two major layers:

1. *Initialisation.* Involves the *Buffer* and the *Authorised_User*, and using channel *push*.

2. *Operations.* This involves the *Buffer* and any other user.

$$Init \;\widehat{=}\; push\,!\,v \;\;\|\;\; (push\,?\,x \;;\; empty := \texttt{false}) \;\|\; \texttt{skip}$$

2. Layer Design.

(5.1)

$$Operation \;\widehat{=}\; \left(\texttt{skip} \;\|\; \left[\begin{array}{l} \texttt{while true do} \\ \quad \{ \\ \quad\quad empty \;\rightarrow\; push\,?\,x \\ \quad \Box \\ \quad\quad \texttt{not } empty \;\rightarrow\; pull\,!\,v \\ \quad \} \\ \quad\quad\quad \texttt{od} \end{array} \right] \;\|\; \left[\begin{array}{l} pull\,?\,x\,; \\ push\,!\,v \end{array} \right] \right)$$

It is clear that each of the above layers are communication-closed and the resulting quasi-sequential system is

$$Sys_L \;\widehat{=}\; Init \;;\; Operation \;\cdots\cdots\cdots\; (a)$$

3. Integration. In this phase, the layers are integrated to obtain the final system:

$$Sys \;\widehat{=}\; Authorised\_User \;\|\; Buffer \;\|\; User \;\cdots\cdots\cdots\; (b)$$

where

$$
(5.2) \qquad \mathit{Buffer} \mathrel{\widehat{=}} \left(\begin{array}{l}
\mathit{push}\,?\,x\,; \\
\mathit{empty} := \texttt{false}\,; \\
\texttt{while true} \\
\quad \texttt{do} \\
\qquad \{ \\
\qquad\quad \mathit{empty} \;\rightarrow\; \mathit{push}\,?\,x \\
\qquad\quad \Box \\
\qquad\quad \texttt{not}\ \mathit{empty} \;\rightarrow\; \mathit{pull}\,!\,v\,; \\
\qquad\qquad\qquad\qquad \mathit{empty} := \texttt{true} \\
\qquad \} \\
\quad \texttt{od}
\end{array} \right)
$$

and the users are modelled as

$$
(5.3) \qquad \mathit{Users} :: \left(\begin{array}{lll}
\mathit{Authorised\_User} & \mathrel{\widehat{=}} & \mathit{push}\,!\,v \\
\mathit{User} & \mathrel{\widehat{=}} & \mathit{pull}\,?\,x \;;\; \mathit{push}\,!\,v
\end{array} \right)
$$

We note that, as the layers were designed communication-closed, then $\mathit{Sys\_L} \equiv \mathit{Sys}$.

Further Layering and Analysis. The above *Operation* layer can be further decomposed by unravellings both the loop and non-deterministic choice sub-structures. Let

$$
(5.4) \qquad Buf \;\widehat{=}\; \left(\begin{array}{l}
\texttt{while true} \\
\quad \texttt{do} \\
\qquad \{ \\
\qquad\quad \mathit{empty} \;\rightarrow\; \mathit{push}\,?\,x \\
\qquad\quad \Box \\
\qquad\quad \texttt{not}\ \mathit{empty} \;\rightarrow\; \mathit{pull}\,!\,v\,; \\
\qquad\qquad\qquad\qquad \mathit{empty} := \texttt{true} \\
\qquad \} \\
\quad \texttt{od}
\end{array}\right)
$$

And let us first consider the loop's body

$$
(5.5) \qquad Buf_{body} \;\widehat{=}\; \left(\begin{array}{l}
\mathit{empty} \;\rightarrow\; \mathit{push}\,?\,x \\
\Box \\
\texttt{not}\ \mathit{empty} \;\rightarrow\; \mathit{pull}\,!\,v\,; \\
\qquad\qquad\qquad\qquad \mathit{empty} := \texttt{true}
\end{array}\right)
$$

Using the process in Section(5.3) (page 170), Buf_{body} is semantically equivalent to

$$\texttt{true} \rightarrow \left\{ \begin{array}{l} \left[\begin{array}{l} \mathit{empty} \rightarrow (\mathit{push}\ ?\ x) \\ \Box \\ \texttt{not}\ \mathit{empty} \rightarrow \texttt{skip} \end{array} \right] \\ \quad ; \\ \left[\begin{array}{l} \texttt{not}\ \mathit{empty} \rightarrow ((\mathit{pull}\ !\ v)\ ;\ \mathit{empty} := \texttt{true}) \\ \Box \\ \texttt{not}\ \mathit{empty}\ \texttt{and}\ \mathit{empty} \rightarrow \texttt{abort} \\ \Box \\ \mathit{empty} \rightarrow \texttt{skip} \end{array} \right] \end{array} \right\}$$

$\Box$

$$\texttt{true} \rightarrow \left\{ \begin{array}{l} \left[\begin{array}{l} \texttt{not}\ \mathit{empty} \rightarrow (\mathit{pull}\ !\ v)\ ;\ \mathit{empty} := \texttt{true} \\ \Box \\ \mathit{empty} \rightarrow \texttt{skip} \end{array} \right] \\ \quad ; \\ \left[\begin{array}{l} \mathit{empty} \rightarrow (\mathit{push}\ ?\ x) \\ \Box \\ \mathit{empty}\ \texttt{and}\ \texttt{not}\ \mathit{empty} \rightarrow \texttt{abort} \\ \Box \\ \texttt{not}\ \mathit{empty} \rightarrow \texttt{skip} \end{array} \right] \end{array} \right\}$$

Using the *Alt* algebraic laws ($Alt-1-Alt-4$), Buf_{body} can be reduced to

$$\texttt{true} \rightarrow \left\{ \begin{array}{l} \left[\begin{array}{l} \mathit{empty} \rightarrow (\mathit{push}\ ?\ x) \\ \Box \\ \texttt{not}\ \mathit{empty} \rightarrow \texttt{skip} \end{array} \right] \\ \quad ; \\ \left[\begin{array}{l} \texttt{not}\ \mathit{empty} \rightarrow ((\mathit{pull}\ !\ v)\ ;\ \mathit{empty} := \texttt{true}) \\ \Box \\ \mathit{empty} \rightarrow \texttt{skip} \end{array} \right] \end{array} \right\}$$

$\Box$

$$\texttt{true} \rightarrow \left\{ \begin{array}{l} \left[\begin{array}{l} \texttt{not}\ \mathit{empty} \rightarrow (\mathit{pull}\ !\ v)\ ;\ \mathit{empty} := \texttt{true} \\ \square \\ \mathit{empty} \rightarrow \texttt{skip} \end{array} \right] \\ \left[\begin{array}{l} \mathit{empty} \rightarrow (\mathit{push}\ ?\ x) \\ \square \\ \texttt{not}\ \mathit{empty} \rightarrow \texttt{skip} \end{array} \right] \end{array} \right\} ;$$

And further, Buf_{body} is transformed to

$$Buf_{body} \mathrel{\widehat{=}}$$

$$\begin{array}{l} ((\ \texttt{not}\ \mathit{empty} \rightarrow \mathit{pull}\ !\ v\ ;\ \mathit{empty} := \texttt{true}\ )\ \square\ (\ \mathit{empty} \rightarrow \mathit{push}\ ?\ x\ ))\ ; \\ \quad \left[\begin{array}{l} \mathit{empty} \rightarrow \mathit{push}\ ?\ x \\ \square \\ \texttt{not}\ \mathit{empty} \rightarrow \texttt{skip} \end{array} \right] ; \left[\begin{array}{l} \texttt{not}\ \mathit{empty} \rightarrow \mathit{pull}\ !\ v\ ;\ \mathit{empty} := \texttt{true} \\ \square \\ \mathit{empty} \rightarrow \texttt{skip} \end{array} \right] ; \\ ((\ \texttt{not}\ \mathit{empty} \rightarrow \mathit{pull}\ !\ v\ ;\ \mathit{empty} := \texttt{true}\ )\ \square\ (\ \mathit{empty} \rightarrow \mathit{push}\ ?\ x\ )) \end{array}$$

We note that

$$Buf \mathrel{\widehat{=}} Buf_{body}\ ;\ Buf$$

Now are ready to construct some further layers. For example, we can readily have two layers, *PULL*, *SKIP* and *PUSH*:

$$PULL \mathrel{\widehat{=}} \left(\begin{array}{l} \texttt{not}\ \mathit{empty} \rightarrow \mathit{pull}\ !\ v\ ;\ \mathit{empty} := \texttt{true} \\ \square \\ \mathit{empty} \rightarrow \mathit{push}\ ?\ x \\ \| \\ \mathit{pull}\ ?\ x \end{array} \right) \tag{5.6}$$

and

$$PUSH \;\widehat{=}\; \left(\begin{array}{l} empty \;\rightarrow\; push\,?\,x \\ \Box \\ \texttt{not}\; empty \;\rightarrow\; \texttt{skip} \end{array} \right) \;\|\; \left(\begin{array}{l} \\ \\ push\,!\,v \end{array} \right) \tag{5.7}$$

and

$$SKIP \;\widehat{=}\; \texttt{skip} \,\|\, \left(\left[\begin{array}{l} \texttt{not}\; empty \;\rightarrow\; pull\,!\,v \;;\; empty := \texttt{true} \\ \Box \\ empty \;\rightarrow\; \texttt{skip} \end{array} \right] \;\; \left[\begin{array}{l} \texttt{not}\; empty \;\rightarrow\; pull\,!\,v \;;\; empty := \texttt{true} \\ \Box \\ empty \;\rightarrow\; push\,?\,x \end{array} \right] \;\; ; \right)$$

The *Operation* layer in equation (a) (page 178) can now be unraveled to become:

$$(PULL \;;\; PUSH \;;\; SKIP) \;;\; Buff$$

We should note that the above example has only one user, the generalisation to more than one user is straightforward.

5.5 Fault-Tolerance Provisions

"Atomicity" and "atomic actions" [18, 66, 72, 87, 88] have been recognised as important concept, both at specification, design and implementation of computing systems. They were first introduced as a means to characterise programs by their input and output relations (i.e. transformation systems) and many error recovery techniques have been introduced [18, 66, 72].

The interest in devising error recovery techniques has increased due to the fact that it may be generalised to similar concepts to recovery when dealing with parallel systems. As it turned out, error recovery in parallel systems were much harder.

As for workflow systems, which are

- inherently distributed;
- may contain, what is know to be, *long-running* activities; and are also
- security-critical,

this conventional failure model must be re-considered as designers need to deal with *partial computations*. These partial computations, may have accessed and/or altered sensitive information.

Atomic activity, or *atomicity* in general, is a portion of the program that enjoys primitive status with regards to its environment, though it may have a rather complicated internal structure and processing. For example, in database setting, this

has been widely used and acknowledged extensively and are known a *transactions*.

The characterisation of atomicity is sometimes known as *serialisability* property: *an activity is known to be atomic if it can be considered, as far as its environment is concerned, to be indivisible and instantaneous, [43].* Once this is understood, atomic activities allow recoverability in a simple manner. In what follows, we show how layers are in fact atomic actions which facilitate the design of fault-tolerant workflows.

5.6 Layers and Atomicity

The layer structure presented here, offer a general framework for structuring, building and designing fault-tolerant workflows systems. The concept of *communication-closed* layer provides a mechanisms to decompose any given workflow system into a set of a "basic" atomic actions. Furthermore, as a result of this decomposition, the new quasi-sequential flow may be looked upon as a "planned" systems which is designed explicitly as a set of "planned atomic activities".

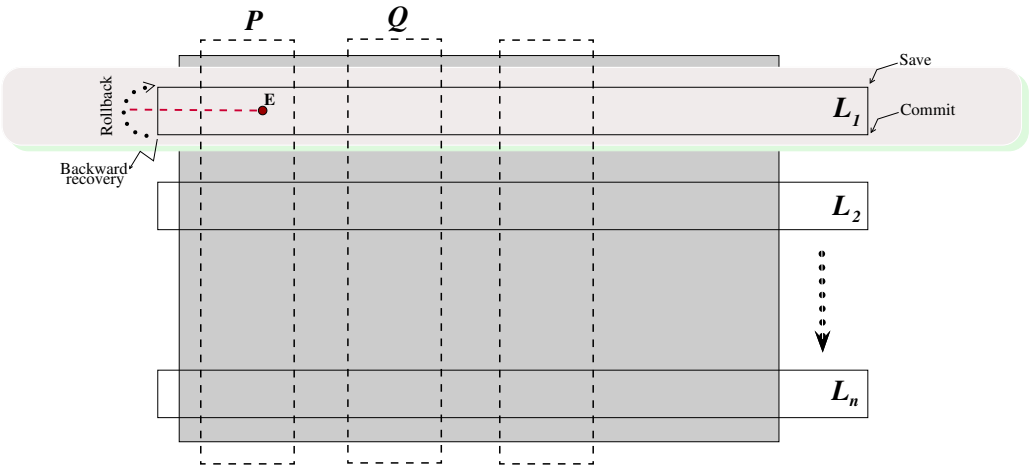


Figure 5.2 – Layer's Atomicity Structure

A planned system is often referred to as "atomic systems". We should note here that the layer structure does not allow for what is known as "spontaneous" atomic actions that arise the dynamic sequences of events occurring in the system - i.e. behavioures. This is because the property of communication closeness layer delimits any error propagation caused by interprocess communication. It also support the idea of error conferment.

5.6.1 Error Recovery in the Layer Structure

It is well known that an error in a workflow corresponds to an inconsistent behaviour of its constituents activities. These behaviours may take the forms of unexpected termination or an erroneous communication attempt. It should be realised that if computation is successful, provisions for fault tolerance within a

layer are invisible to the rest of the layers. This leads to encapsulation of such measures in a rather modular fashion (we call this sub-layering).

Additionally, and as pointed out in [86], the specification of a layer is constituted by the relationship between the states at the beginning and the termination of the layer.

5.6.2 Recovery Procedure

Let us assume that an error, E, has occurred at a layer L_i. A typical recovery procedure may be informally described as follows: For each layer there is what shall be called "*Layer-Handler*". A "*Layer-Handler*" acts as a local handler for this layer (as depicted in Fig(5.3)).

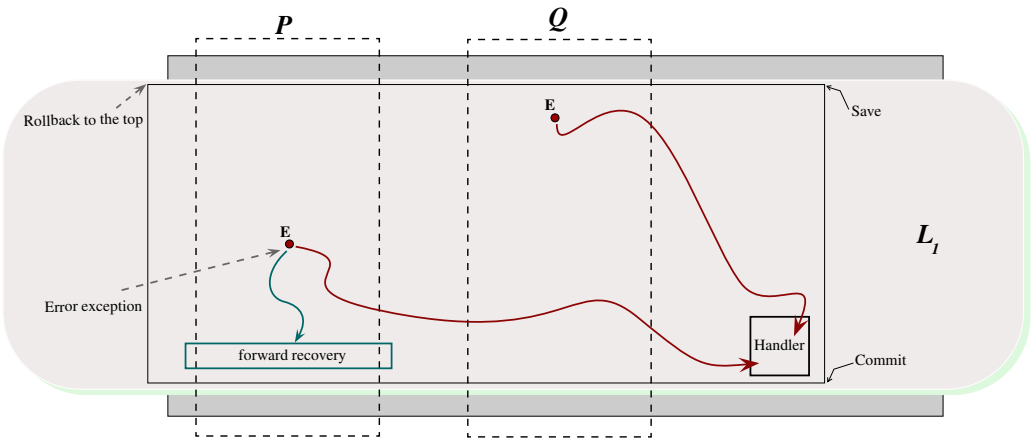


Figure 5.3 – Layer with Error Handlers

At the boundary of a layer, say L_i and before the execution resumes, it is assumed that all states are saved. We also assume that a local handler, H_i, exists for each

layer and that it has the necessary software that can be invoked to attempt correcting E. If this attempt is successful, then the flow will resume at the subsequent layer, i.e., L_{i+1}, otherwise the computation is rolled back to a previously saved consistent state. However, if a failure persists to occurs after the resumption in all layers above, then the whole application must be aborted. Aborting computation occur only when failure happens at the most outer layer. Further, in case of nested layers, the following philosophy for error recovery is adopted. Any internal layer should have its own private ("local") handler which deal with any exception raised by any of its threads. If the exception can not be handled internally, then it must be raised in its containing layer (this is known as "layer-failure") and by *all* of its threads. However if different exceptions are raised by more than one thread of a layer and at least one can not be handled locally, a layer-failure should be raised.

5.6.3 Backward Error Recovery

Backward error recovery techniques can be realised within our layer structure. We closely follow established mechanisms that are reported in Liskov [87, 88] and Baiardi [18]. In our layer structure, the top boundary of every layer acts as a *recovery line* for backward error recovery mechanisms. As we mentioned above, as control enters the recovery line, all states are saved [5]. Therefore in case of an error, a mechanism must be establish to undo the whole layer with, for example,

[5] This resembles a stable storage techniques.

the invocation of an alternative method for that particular layer. Techniques such as N-Version programming may be adopted.

Note that the saved state at the top boundary of a layer can, in case of errors, be restored before the resumption of recovery mechanisms. We can achieve this by parametrising the layer itself. The parameters play a similar role to the formal parameter list in a procedure/function/method environment.

5.6.4 Forward Error Recovery

For forward error recovery we aim to remove or isolate a specific fault. Obviously, this requires the correct identification of the fault and the erroneous state of the layer which has been corrupted before proceeding with further processing. It has become a common practice to use exception handling mechanisms in dealing with these types of errors. The local handler (which is a sub-layer) cope with all exceptions raised in this layer. A classification of errors should be identified and for each class of error, there should be an exception value which once raised by any activity then all other activities are made aware of the exception. Otherwise, if no exceptions are raised, the layer is considered to terminate successfully.

5.7 Summary

Motivated by the fact that workflows can be highly distributed with timing, security and context constraints, we have developed a design language, $\mathcal{CS}-\mathcal{F}low$, where "context" is a first class citizen and that "guards" provide expressive tool for security and context requirements. This has obviated the need to have extra and specialised language to express security policies. The language was given a specification-oriented semantics in a process algebraic style. We have also given equational characterisation to the language. Such algebraic laws are useful tools to manipulate, at a textual level, $\mathcal{CS}-\mathcal{F}low$ systems.

Given this background, it was important to devise a methodology for the development and analysis of $\mathcal{CS}-\mathcal{F}low$ workflow. For this, we have introduced the concept of "communication-closed" layers which is a logical structure that transform a given $\mathcal{CS}-\mathcal{F}low$ system into a semantically equivalent one which is easier to analyse. The resulting system is quasi-sequential and is a composition of a number of communication-closed layers. For a layer to be communication-closed, a communication must start and terminates within the layer itself. In other words, communications do not cross the boundary of layers. The composition of layers could be purely sequential, $;_1^n \{L_i\}$, or purely in a non-deterministic fashion, $\Box_1^n \{G_i \rightarrow L_i\}$. Once a layer is constructed, we can apply various algebraic laws to resolve, e.g. communication and interference that render the analysis easier.

Having introduced the layer concept as an analysis tool, we have devised a development methodology of $\mathcal{CS}-\mathcal{F}low$ system. In such a methodology, we decompose requirements into layer-requirements, develop each layer then integrate layers to form the traditional and conventional $\mathcal{CS}-\mathcal{F}low$ system. As part of future work, we could explore the idea having a layer as a syntactic construct and some static analyses to be done at compilation time.

As timing issues are central to many workflows, we have argued that communication-closed layer can be extended to become "time-closed" layer where time constrains the way we construct the layers themselves. A layer is called "closed" if and only if it is both time- and communication-closed.

We have ended the chapter by elaborating on how the layer design can serve as a mechanism for developing fault-tolerant $\mathcal{CS}-\mathcal{F}low$ system. This is because of atomicity property that we get for free from the fact layers are communication-closed. To this end, backward and forward error recovery techniques can be deployed for each layer.

Next chapter we deal with tool support that enables us to execute/animate $\mathcal{CS}-\mathcal{F}low$ system. The tool only serves as a proof of concept.

Chapter 6

$\mathcal{CS}-\mathcal{F}low$

ANIMATION and VALIDATION

Objectives

- Provide reduction rules for $\mathcal{CS}-\mathcal{F}low$ activities.
- Process Algebraic-Style Semantics for $\mathcal{CS}-\mathcal{F}low$.
- Provide a description for practical support for $\mathcal{CS}-\mathcal{F}low$

6.1 Introduction

In this chapter we describe the reduction rules for $\mathcal{CS}-\mathcal{F}low$ activities. Reduction rules describe how an activity can be executed in a step-by-step fashion and

are used as the basis for validating (animating) by exploring various scenarios of designs/models of $\mathcal{CS}-\mathcal{F}low$ system.

Being a process algebra, *computation* is not possible within CCA. The focus there was on the animation of context, context-awareness and mobility. Indeed, simple operations such as $\leq$, $\geq$, etc, are not supported in its execution environment. Therefore, the CCA execution environment was modified to include variables, usual arithmetic operations, timing issues as well as priorities constructs. In this chapter we give an encoding of $\mathcal{CS}-\mathcal{F}low$ to be suitable of execution together with few illustrative examples. The chapter is organised as follows. In Section (6.2), we give the reduction rules for $\mathcal{CS}-\mathcal{F}low$, the encoding process of $\mathcal{CS}-\mathcal{F}low$ is given in Section (6.3), with few illustrative examples in Section (6.3). In Sections (6.3.6) and (6.3.7), two directives to control the execution of $\mathcal{CS}-\mathcal{F}low$ programs are described. Finally, some examples of executions are shown in Section(6.4).

6.2 $\mathcal{CS}-\mathcal{F}low$ Reduction Rules

Reduction rules serve two purposes. Rules allows us to understand how an activity within a workflow are executed; and the operational semantics of any language is often defined using a structural congruence $\equiv$ and a reduction relation $\rightarrow$. The structural congruence for $\mathcal{CS}-\mathcal{F}low$ was given as an algebraic characterisation

Table 6.1 – Reduction Rules-1

$P \rightarrow P'$	$\Rightarrow$	$\texttt{var}\ \tilde{x}\ \texttt{in}\ \{P\} \rightarrow \texttt{var}\ \tilde{x}\ \texttt{in}\ \{P'\}$	(Reduction Var)
$P \rightarrow P'$	$\Rightarrow$	$\texttt{chan}\ \tilde{x}\ \texttt{in}\ \{P\} \rightarrow \texttt{chan}\ \tilde{x}\ \texttt{in}\ \{P'\}$	(Reduction Chan)
$P \rightarrow P'$	$\Rightarrow$	$\alpha\ <\tilde{x}>:\{P\} \rightarrow \alpha\ <\tilde{x}>:\{P'\}$	(Reduction Contxt)
$P \rightarrow P'$	$\Rightarrow$	$\mathcal{C}(P) \rightarrow \mathcal{C}(P')$	(Reduction Contxt)
$P \rightarrow P'$	$\Rightarrow$	$P \parallel Q \rightarrow P' \parallel Q$	(Reduction Par)
$P \equiv Q,\ Q \rightarrow Q',\ Q' \equiv P'$	$\Rightarrow$	$P \rightarrow P'$	(Reduction $\equiv$)

in Chapter(4) (page 119). These laws also allow us to manipulate the structure of activities at textual level, i.e. it can be the bases of a term re-writing tool.

We use the customary notation $P\{\tilde{y} \leftarrow \tilde{z}\}$ for the substitution of each name in the list $\tilde{z}$ for each free occurrence of the corresponding name in the list $\tilde{y}$. Obviously, such a substitution is only defined if $\mid \tilde{y} \mid = \mid \tilde{z} \mid$. The reduction relation of activities is defined in Table(6.1) and Table(6.2).

6.3 Encoding $\mathcal{CS}-\mathcal{F}low$ Activities

The transformation from $\mathcal{CS}-\mathcal{F}low$ to ccaPL is given below.

Table 6.2 – Reduction Rules-2

$(Chan\ ?\ \tilde{y})\ ;\ P \parallel (Chan\ !\ \tilde{z})\ ;\ Q$	
$\rightarrow\ \ P\{\tilde{y} \leftarrow \tilde{z}\} \parallel Q$	(Reduction Com-1)
$\alpha : \{((Chan\ ?\ \tilde{y})\ ;\ P) \parallel Q\}\ \parallel\ \beta : \{((Chan\ !\ \tilde{z})\ ;\ R) \parallel S\}$	
$\rightarrow\ \ \alpha : \{P(\tilde{y} \leftarrow \tilde{z}) \parallel Q\} \parallel \beta : \{R \parallel S\}$	(Reduction Com-2)
$\alpha : ((Chan\ ?\ \tilde{y})\ ;\ P) \parallel Q) \parallel \beta : (\alpha : (Chan\ !\ \tilde{z})\ ;\ R) \parallel S)$	
$\rightarrow\ \ \alpha : (P(\tilde{y}\ \leftarrow\ \tilde{z})) \parallel \beta : (R \parallel S)$	(Reduction Com-3)
$\alpha : (\beta : (Chan\ ?\ \tilde{y}\ ;\ P) \parallel Q) \parallel \beta : (Chan\ !\ \tilde{z}\ ;\ R) \parallel S$	
$\rightarrow\ \ \alpha : (P(\tilde{y}\ \leftarrow\ \tilde{z}) \parallel Q) \parallel \beta(R \parallel S)$	(Reduction Com-4)
$\alpha : (\beta : (Chan\ ?\ \tilde{y}\ ;\ P) \parallel Q) \parallel \beta : (\alpha : ((Chan\ !\ \tilde{z}\ ;\ R \parallel S))$	
$\rightarrow\ \ \alpha : (P(\tilde{y}\ \leftarrow\ \tilde{z}) \parallel Q) \parallel \beta(R \parallel S)$	(Reduction Com-5)
$\beta :\ \{\mathtt{to}(\alpha)\ .\ P \parallel Q\}\ \parallel\ (\alpha :\ \{R\})$	
$\rightarrow\ \ \alpha :\ \{\beta : \{P \parallel Q\}\ R\}$	(Reduction Mob)

6.3.1 Variables

A variable x in $\mathcal{CS}-\mathcal{F}low$ will be treated as a memory cell and is modelled by the following process abstract where v is the initial value of the variable x:

```
proc mem(x, v) {
   x[
        send(v).0
       | !@recv().recv(w).{ @send(w).0 | send(w).0 }
       | !@recv(u).recv(y).{ @send().0 | send(u).0 }
    ]
}
```

To read the value of a variable x into the the name n use the process:

```
x#send().x#recv(n)
```

To write a value t in a variable x, use the process:

```
x#send(t).x#recv()
```

6.3.2 Channels

A channel c is modelled as an ambient of name c. Communication over channels is modelled below. In what follows we show how both primitive and compound activities be modelled.

6.3.3 Primitive Activities

Here we give the encoding of the primitive activities of $\mathcal{CS}-\mathcal{F}low$.

$\texttt{skip} \mathrel{\widehat{=}} \mathbf{0}$

$x := v \;;\; P \mathrel{\widehat{=}} \texttt{x\#send(v).x\#recv()}.P$

$c \,!\, v \;;\; P \mathrel{\widehat{=}} \texttt{c[@send(v).0]}\,.\,P$

$c \,?\, v \;;\; P \mathrel{\widehat{=}} \texttt{c\#recv(v).del c}.P$

$\alpha : \{P\} \;\widehat{=}\; \alpha\,[P]$

$\texttt{to}(\alpha);\, P \;\widehat{=}\;$ `new m {`

`m[@send().0]`

`| <somewhere (`α`[this | true] | true)>m#recv().del m.out.`P

`| in `α`.m#recv().del m.`P

`}`

$\texttt{var}\; x \;\texttt{in}\; P \;\widehat{=}\;$ `new x {` P `| mem(x,0) }`

$\texttt{chan}\; c \;\texttt{in}\; P \;\widehat{=}\;$ `new c` P

6.3.4 Compound Activities

The encoding of the compound activities are as follows.

$P \parallel Q \mathrel{\widehat{=}} P \ \texttt{|}\ Q$

```
while G.P ≙ new x {
                x[@send().0]
              | <not G>x#recv().del x.0
              | ! <G>x#recv().del x.P.x[@send().0]
          }
```

$p_1 : G_1 \to P \,\square\, p_2 : G_2 \to P \mathrel{\widehat{=}}$

```
new x {
                    x[@send().0]
              | <(G1 and not G2) or (
                    G1 and G2 and p1>=p2)>x#recv().del x.P
              | <(G2 and not G1) or (
                    G2 and G1 and p2>=p1)>x#recv().del x.Q
          }
```

```
delay(t); P ≙ new clock new m {

                  clock[send(0).0 | !@recv().recv(w).

                     {@send(w).0 | send(w+1).0}]

               | m[@send().0]

               | !m#recv().clock#recv().clock#recv(x).{

                           <x=t>del m.P

                        | <not(x=t)>del m.m[@send().0]

                     }

            }
```

6.3.5 Encoding the guards

$\texttt{true} \mathrel{\widehat{=}} \texttt{true}$

$\neg\, E \mathrel{\widehat{=}} \texttt{not}\, E$

$E_1 \wedge E_2 \mathrel{\widehat{=}} E_1\, \texttt{and}\, E_2$

$E_1 \vee E_2 \mathrel{\widehat{=}} E_1 \texttt{or}\, E_2$

$\texttt{somewhere}(\alpha).E \mathrel{\widehat{=}} \Diamond\, \alpha\, E$

6.3.6 Execution directives

As we mentioned above, there are two directives to control the execution of $\mathcal{CS}-\mathcal{F}low$ programs: `mode` and `display`.

1. *M*ode: This handles the way concurrent and non-deterministic activities are executed. As activities in a model are executed concurrently in an interleaving manner, the directive `mode` specifies the strategy for fair execution of these activities [1]. `mode` takes a unique parameter `random`: `mode random`.

   With `random`, the activity to be executed is chosen randomly from the list of enabled activities. A pseudo random number generator with uniform distribution is assumed.

   However, when this directive is not specified, the default execution mode is assumed (*d*eterministically): i.e. at each execution step the activity to execute is chosen based on two criteria:

   - how long the activity has been willing to execute (first-in, first-out), otherwise,
   - in case of conflict, sequential order of the text is used.

2. *D*isplay: This directive determines how the execution trace is displayed:

   - `display code` — Displays the model (without comments) after each reduction.

[1] Activity fairness means that if an activity is allowed to be executed infinitely often then that activity is also executed infinitely often. In other words, an enabled activity cannot wait indefinitely to be executed. So each activity must be given a fair chance of being executed.

- `display congruence` — Displays, in addition to reduction steps (denoted by `-->`), the congruence steps (denoted by `<-->`).
- By default code and congruence are not displayed in output, only reduction steps are.

6.3.7 Notations

In general, the following notations are used:

- The symbol '`<-->`' corresponds to the structural congruence relation as formally defined in Chapter 4 (page 119).
- '`-->`' represents the reduction relation (see Table (6.2) and (6.1))
- The explanation of each transition is given between a pair of curly brackets.
- The notation '`A ==X==> B`' means that a context '`A`' has sent a message '`X`' to another context '`B`'.
- Additional annotations such as '`Child to parent`' and '`Sibling to sibling`' provide information about the relationship between the sender '`A`' and the receiver '`B`'. This is only available if the user choose to put structure on contexts.
- The notation `{binding: n -> IN1}` corresponds to the execution of a activity of the form "`find n:k for ...`" and means that the variable *n* has

been bound to the name `IN1`.

6.4 Interface and Examples

As a proof-of-concept, we have built a simple environment within which we can validate/animate our specification/design. As customary, the environment consists of an interface and an execution engine.

The interface is simple and offers basic features such as *file, Edit, etc.* and linked with the execution environment which is responsible for executing/animating $\mathcal{CS}-\mathcal{F}low$ design. In Figures((6.1), (6.2) & (6.3)), illustrates such an interface through some screen shots.

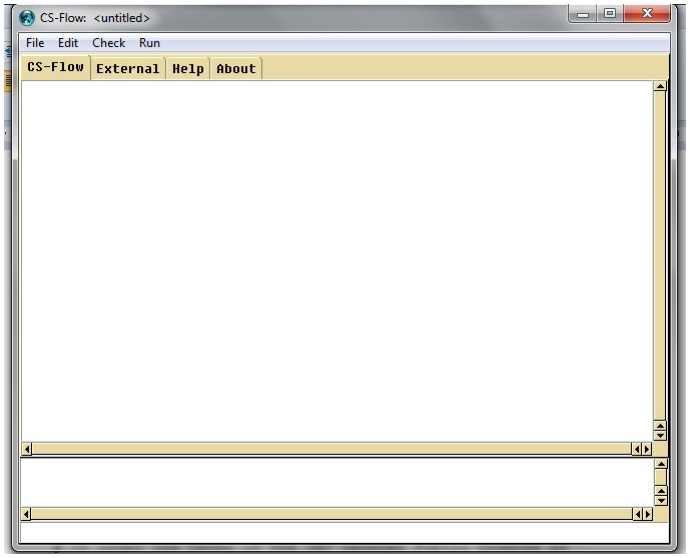


Figure 6.1 – $\mathcal{CS}-\mathcal{F}low$: Front Screen

This interface allows us the normal operations: open/edit a file and be ready to execute as shown below in Figure(6.2).

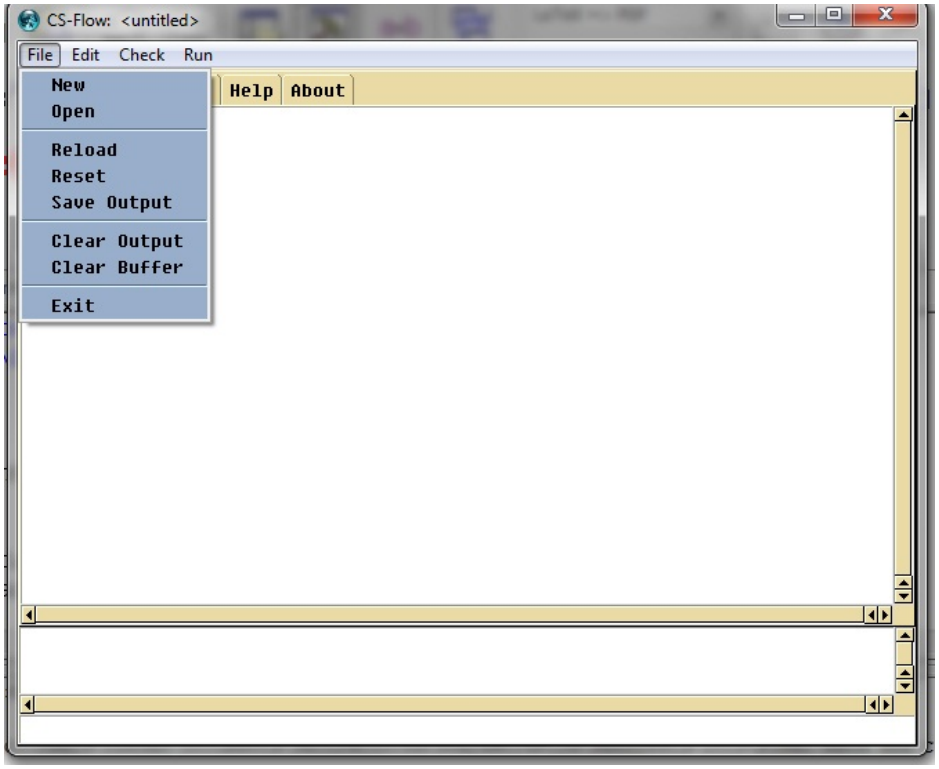


Figure 6.2 – $\mathcal{CS}-\mathcal{F}low$: Menues

Figure(6.3) shows an example.

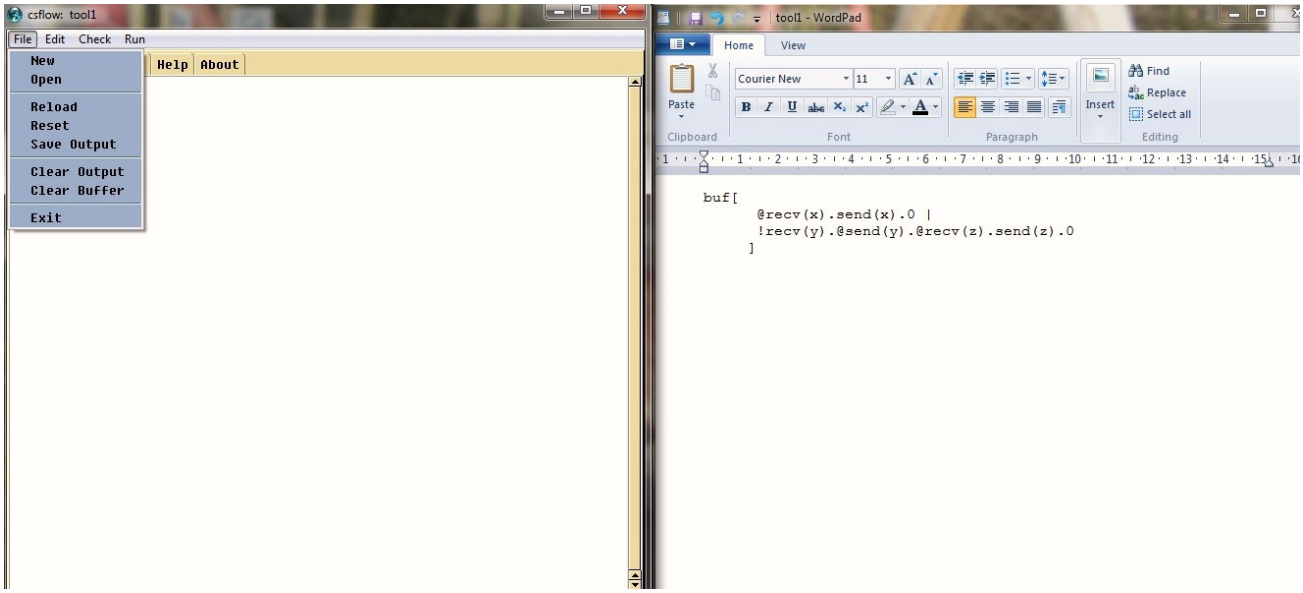


Figure 6.3 – $\mathcal{CS}-\mathcal{F}low$: Editing

The following illustrates the execution environment using some of our working examples.

6.4.1 Example 1: a One-Place Buffer

Let us recall the one-place buffer example given in Section(5.4.1). We recall that a one-place buffer is a data structure that operates through two actions *push* and *pull*, respectively putting one item in the buffer and taking one item from it. The buffer is full when it contains one item. It is impossible to put one item into a full buffer; it is impossible to take one item from the empty buffer.

$\mathcal{CS}-\mathcal{F}low$ model is first presented and followed by its execution form and animation.

```
Buffer ≙
        push ? x ;
        empty := false ;
        while true
          do
              {
                   empty  →  push ? x
                   □
                   not empty  →  pull ! v ;
                                 empty := true
              }
          od
```

```
BEGIN_DECLS

END_DECLS

proc buffer(buf){
   buf[@recv(x).send(x).0 | !recv(y).@send(y).@recv(z).send(z).0]
}

//this process pushes the value v onto the buffer buf.

| proc push(v, buf){
   buf#send(v)
}

| proc pull(buf){
   buf#recv(x).send(x).0
}
 | buffer(A) // create a buffer named A
 | buffer(B) // create a buffer named B
 | push(8,A) // push the value 8 onto the buffer A
 | pull(A)  // pull the value in the buffer A
 | recv(t).  // get the value in the variable t
   push(t,B).0 // and push that value onto the buffer B.
```

```
   ****************************************
   **   csFlow Interpreter version 2.7     **
   **         April 2012                   **
   ****************************************
Execution mode: interleaving
---> {local call to the abstraction "buffer" in the context "root"}
---> {local call to the abstraction "buffer" in the context "root"}
---> {local call to the abstraction "push" in the context "root"}
---> {local call to the abstraction "pull" in the context "root"}
---> {Parent to child: root ===(8)===> A}
---> {Local: A ===(8)===> A}
---> {Child to parent: A ===(8)===> root}
---> {Local: root ===(8)===> root}
---> {local call to the abstraction "push" in the context "root"}
---> {Parent to child: root ===(8)===> B}
---> {Local: B ===(8)===> B}
```

6.4.2 Example 2: Adaptable activities On Shop-floor

We recall the "Adaptable Activities" discussed in Section(3.5.2), page 105. In the scenario, a context-aware system that enables a mobile software agent to edit/view a text/image file on any host device using an appropriate text/image editor for that device's operating system. Here we repeat a fraction of the specification to ease readability: Let's use *win* and *Linx* to denote context running Windows and Linux, respectively. Each of these contexts contains an activity abstraction *edit* that maps to the default text editor and the mobile agent just has to call that activity abstraction to edit a file using the local text editor as specified below. Our *ShopFloor* is specified as a context which has two devices: *win* and *linx*:

$$
\begin{array}{l}
\mathit{ShopFloor}: \\
\quad \{ \\
\qquad \mathit{win} \parallel \mathit{linx} \\
\quad \}
\end{array}
$$

The Linux device is specified as

$$
\begin{array}{l}
\mathit{linx}: \\
\quad \{ \\
\qquad \texttt{var}\, f\, \texttt{in}\, \mathit{edit} \\
\qquad\qquad\qquad \{ \\
\qquad\qquad\qquad\quad \mathit{emacs}(f) \\
\qquad\qquad\qquad \} \\
\quad \}
\end{array}
$$

The corresponding execution form when the host is *linx* is given as follows.

```
lin[
     proc edit(f) {
        emacs(f).0
     }
   |
     agent[
        @edit(foo).0
     ]
]
```

6.4.3 Example 3: Mobility

In this example, we have three contexts, A, B, and C. Upon receiving a message, *msg0* from B, context A moves into B and while it is there, it waits until receiving a message, *msg1* from C and then moves there. Once it is there it terminates.

The model is first shown in $\mathcal{CS}-\mathcal{F}low$ and followed by its execution form and animation.

```
BEGIN_DECLS
  mode random
  display code
  display congruence // add this line to include congruence
```

```
END_DECLS


    A[B:: recv(msg0).in B.0]

 |  B[A:: send(msg0) . C:: recv(msg1).in C.0]

 |  C[B:: send(msg1).0]
```

```
Execution mode: interleaving

 BEGIN_DECLS


 END_DECLS

      A[ B::recv(msg0).in B.0 ]
   |
      B[ A::send(msg0).C::recv(msg1).in C.0 ]
   |
      C[ B::send(msg1).0 ]

--->  {Sibling to sibling: B ===(msg0)===> A}

 BEGIN_DECLS


 END_DECLS

      A[ in B.0 ]
   |
      B[ C::recv(msg1).in C.0 ]
   |
      C[ B::send(msg1).0 ]

--->  {ambient "A" moves into ambient "B"}

 BEGIN_DECLS


 END_DECLS

      B[
            A[ 0 ]
         |  C::recv(msg1).in C.0
      ]
   |
      C[ B::send(msg1).0 ]

--->  {Sibling to sibling: C ===(msg1)===> B}

 BEGIN_DECLS


 END_DECLS

      B[
            A[ 0 ]
         |  in C.0
      ]
```

```
Execution mode: random

 BEGIN_DECLS


 END_DECLS

      A[ B::recv(msg0).in B.0 ]
   |
      B[ A::send(msg0).C::recv(msg1).in C.0 ]
   |
      C[ B::send(msg1).0 ]

--->  {Sibling to sibling: B ===(msg0)===> A}

 BEGIN_DECLS


 END_DECLS

      A[ in B.0 ]
   |
      B[ C::recv(msg1).in C.0 ]
   |
      C[ B::send(msg1).0 ]

--->  {ambient "A" moves into ambient "B"}

 BEGIN_DECLS


 END_DECLS

      B[
            A[ 0 ]
         |  C::recv(msg1).in C.0
      ]
   |
      C[ B::send(msg1).0 ]

--->  {Sibling to sibling: C ===(msg1)===> B}

 BEGIN_DECLS


 END_DECLS

      B[
            A[ 0 ]
         |  in C.0
      ]
   |
```

6.4.4 Example 4: Variable declaration

The following example demonstrates the declaration of variables and how they are assigned values:

```
/*
test example
display code
*/
BEGIN_DECLS
display code
END_DECLS



proc mem(x, v) {
x[
send(v).0
| !@recv().recv(w).{ @send(w).0 | send(w).0 }
| !@recv(u).recv(y).{ @send().0 | send(u).0 }
]
}
```

```
|

mem(myVar, 200).0

|

myVar#send(1500).myVar#recv().

myVar#send().myVar#recv(n).myValueIs(n).0

//|

//x#send(t).x#recv()
```

```
Execution mode: interleaving

 BEGIN_DECLS


 END_DECLS

      proc mem(x,v) {
            x[
                  send(v).0
               |  ! @recv().recv(w).{
                                 @send(w).0
                              |  send(w).0
                           }
               |  ! @recv(u).recv(y).{
                                 @send().0
                              |  send(u).0
                           }
            ]
      }
   |  mem(myVar,200).0
   |
myVar#send(1500).myVar#recv().myVar#send().myVar#recv(n).myValueIs(n).
0

--->  {local call to the abstraction "mem" in the ambient "root"}

 BEGIN_DECLS


 END_DECLS

      myVar[
            send(200).0
         |  ! @recv().recv(w).{
                           @send(w).0
                        |  send(w).0
                     }
         |  ! @recv(u).recv(y).{
                           @send().0
                        |  send(u).0
                     }
      ]
   |
      proc mem(x,v) {
            x[
                  send(v).0
               |  ! @recv().recv(w).{
                                 @send(w).0
                              |  send(w).0
                           }
```

Figure 6.6 – $\mathcal{CS}-\mathcal{F}low$: Variables

6.5 Summary

We have introduced $\mathcal{CS}-\mathcal{F}low$ that overcame the limitations that exist in current workflow languages and which meet the need for the specification, modelling and design of modern workflow system. Namely, $\mathcal{CS}-\mathcal{F}low$ has a provision

- for context-awareness (to meet the pervasive nature of current workflow),
- to deal with security issues and
- to have the ability to explicitly express timing issues which exist in many, if not all current workflow systems.

We have also given a process algebraic-style semantics to $\mathcal{CS}-\mathcal{F}low$. Such semantics is denotational in nature as it describes the behaviour of an activity as a process term in a novel context aware process calculus known as CCA. The choice of CCA, as a base formalism, was due to the fact that it is the closest process algebra that satisfy our requirements. We have also given an algebraic characterisation to $\mathcal{CS}-\mathcal{F}low$ activities and have derived important reduction rule for them The later allows us to execute a model in $\mathcal{CS}-\mathcal{F}low$. The former allows us to manipulate terms in $\mathcal{CS}-\mathcal{F}low$. Further, we have extended the execution environment of CCA to provide a tool support for $\mathcal{CS}-\mathcal{F}low$. In the next chapter we present a realistic case study as an illustration and evaluation to our formalism. This is namely the next generation of Context-Aware (hospital) Ward $\mathcal{CAW}$.

Chapter 7

$\mathcal{CS}-\mathcal{F}low$

THE HEALTH CARE SYSTEM:

The $\mathcal{CAW}$ system

Objectives

- Evaluate our formalism on a realistic case study.
- Evaluate Health care systems workflow of some scenarios.

7.1 Introduction

This chapter deals with the evaluation phase of the work presented here. The evaluation takes two complementary strands:

- animation of $\mathcal{CS}-\mathcal{F}low$'s models and specifications and
- formal verification of time-critical component of workflows.

To facilitate this evaluation, we have chosen a case study from the *health care* domain. Health care provides huge amounts of business workflows, which will benefit from workflow adaptation and support through pervasive computing systems. Several of these workflows have to be performed regularly. For example, we find workflows, such as "Hand-out Medication", "Ordering Drugs", "Review EPRs" and much more. These workflows are central for successfully and efficiently running hospitals and/or nursing homes. The interpersonal relationships between hospital staff (doctors, nurses, support staff, etc) in these organisation are equally crucial and essential in addition to the provision of the daily health care services. These interpersonal relationships require time to both develop and maintain but, more often than not, medical staff's time is consumed by organizational and administrative tasks rather than "care". For that reason there is a great need for improving the day-to-day activities of workflows by utilising current advances in technology. It is also important to note that workflows within health care domain is in fact inherently *security-critical* and *context-aware*. Accessing patient's

record must be done by appropriately authorised staff, the level of access varies according to the role a particular staff is holding (a Sargent or a consultant, for example will have full access to medical records whilst a nurse may not). In addition, as medical records take a variety of forms: textual (e.g., list of medication, description of diagnoses, etc.) and images (e.g., x-rays). These records need to be accessed (and some cases edited) on various devices with different operating systems and applications. In addition, some activities within health care workflows may be time-critical, of a periodic nature and highly distributed across various hospital departments and between different hospitals and non-health care organizations (e.g., insurance companies, financial organisations, law-enforcement departments, etc.). We begin by outlining briefly the current status of a ward in a typical hospital (Section (7.2)). Then in Section (7.3), we describe typical workflow scenarios in any ward (current or next generation). This is followed by a detail $\mathcal{CS}-\mathcal{F}low$ specification/models for the next generation of hospital wards (Section (7.4)).

7.2 Current Status

The current activities for intensive care include the following obvious tasks. A typical hospital ward (or nursing home) has an average number of patients/inhabitants in need of particular services (a realistic population in an average nursing

home is around 100 patients and inhabitants, each with a particular diagnosis). These common every day services consist of activities such as: (a) specific medication needs, (b) personal hygiene, (c) the preparation of meals with focusing on special diets, (c) managing the laundry, etc. There is also a set of trained professionals with different abilities (nurses, care-takers, medics etc.) each with different schedules and working times.

A workflow can be assigned to a patient/inhabitant, trying to fulfill optimal care for him/her, rescheduling appointments (e.g. with a consultant/doctor, therapies, according to the physician orders or other services that satisfy their specific personal needs), and furthermore informing the nursing staff about patient needs. Other workflows can be specifically attached to the nurses, caretakers or physicians. Basically, all tasks in the patient care are performed in interpersonal interaction between the nurse and the patients assigned to this particular nurse. The way the tasks are performed is mainly regulated by a host of quality metrics and targets. There is also a care handbook that contains basic instructions of how a nurse has to deal with a patient to ensure high level quality of treatment and their correctness (i.e. correct medication to the a specified patient/inhabitant).

Until now the documentation, in many places, is performed using a paper-based technique and the belongings of the patients (such as medication, sweets, drinks, etc.) are stored in a basket-system (one basket per person). Improvements in workflows such as for documentation, or automating the documentation of hy-

giene care using bar codes techniques, etc. are very timely and will be very welcomed by staff and patients alike. Whilst improvements were often thought about yet never implemented for a variety of reasons (e.g., financial constraints, policy/decision makers and/or lack of a thorough understanding of its requirements).

In this chapter we do not aim to undertake the specification, modelling and analysis of a complete health care workflow system, but consider only a fraction of it, namely the *medication scenario*. In this scenario there are different types of workflows. Besides the workflows that specifies the act of nursing, such as handing out medications, there are logistic workflows such as ordering drugs.

A reason for choosing the medication scenario is as follows

- it comprises different separate workflows that are linked together and partly depending on each other,

- medication can clearly be related to the act of nursing itself which carries varies access control, authentication and privacy constraints,

- medication (with its workflows) is one clearly defined unit which is not influenced by other workflows outside of medication, and

- the scenario has timing constraints as the scenario is repeated within a fixed time period.

7.3 The composition of the Medication Scenario

As we mentioned earlier, there are a number of workflows that constitute the medication scenario. these are:

- *Drug ordering*,
- *Drug expiry date check*,
- *Preparation for Daily Medication*,
- *Handing out Daily Medication* and
- *Handing out Medication on Demand*

In this section we give a general description of each of the above and this is followed by a detailed specification and modelling of a *context aware ward* where all of these workflows are performed in (Section 7.4).

WF-1: Drug ordering

Drugs are normally stored in labeled trays which are stacked up in air-conditioned storage room whose temperature is carefully monitored. Ordering drugs is a workflow which basically has two complementary activities (*"check stock of drugs"* and *"order drugs"*), which are performed together in one workflow. Checking the stock of drugs is currently performed by a senior nurse who compares the need of specific medicines for the next time

period (until the next check of stock) with the number of drugs exists in its storage tray. If the number of drugs in storage is below the needed drugs in the next time period, the medicine has to be ordered. The calculation has to take into account the ordering and delivery time. In addition, there might be specific constraints for every ward, hospital or care home (e.g. time, day, quantity of orders and specialised pharmacists).

WF-2: Drug expiry date check

It is obviously clear that "Ordering drugs" is activated if there is a shortage in the available drug or some of the existing drugs are out-of-date. Therefore checking drug's expiry date is a workflow that is seen as part of the ordering workflow, as the expiry of drugs influences the stock of drugs and hence the requirement of ordering drugs. The check for expiry date is performed at a regular periodic intervals (in some places, once every month). However, it is not performed within every drug ordering activity. Nevertheless we have to emphasis the dependency of this workflow to the ordering drugs workflow.

Checking the expiry dates are often performed by a senior nurse and an advice from a consultant may be sought.

WF-3: Preparation of daily medications

The preparation of the daily medication (for all patients/inhabitants) is done in the previous night (often by the night nurse). This is because of efficiency

reason, as it will save time during the daily activities of the ward. So, during the day the nurses in the early- and late-shifts do not have to check for all medications and prepare them. Drugs which are not available or out-of-date can not be prepared.

There should be mechanisms and/or procedures in place to ensure the correct preparation of drugs. Human errors are likely to occur (specially at the end of a night shift) and hence a nurse supervisors perform various checks that ensure the correct preparation of drugs.

Again this workflow is part of the whole medication workflow and it depends on the correct drug ordering and checking expiry date workflows.

WF-4: Handing out of daily medications:

The nurse hands out the medication to the patients at breakfast, lunch, dinner and during the night taking into account the patients' needs and the prescriptions of the doctors and/or consultants. This workflow, the handing out of the medication is repeated several times a day. It depends on an efficient preparation of daily medication and on a successful ordering of drugs.

WF-5: Handing out medications on demand

There are many cases where the status of patients/inhabitants has changed and required a different medication which had been prescribed by doctors/-

consultants. In such situation medication has to be handed out on demand. Contrasting this with other medication workflows, this workflow is not performed periodically. It might be executed several times a day or even only once. Again this workflow depends on a successful ordering of drugs, which means that the drugs have to be available on demand. Clearly, it affects and influence the ordering of drugs, as it is difficult to estimate the number of needed on-demand drugs for the next time period. A stock which is too large could cause an unnecessary expiration of the drugs and running out of on-demand medication could possibly cause a crisis. Therefore proper interaction between this workflow and the drug-ordering workflow is very important.

7.4 Next Generation Context-Aware Ward: $\mathcal{CAW}$

As we mentioned earlier, the above workflows are general to both traditional and the next generation wards. In this section, we specify what we call the next generation *Context-aware (hospital) Ward* ($\mathcal{CAW}$); in particular, we consider the workflows: "Handing out Medication" and "Handing out Medication on Demand", given in Section (7.3) and Section (7.3) (page 226). These are partly considered in the *Hospital of the Future* project at the Centre for Pervasive Health Care, [19,20]. This case study was particularly chosen because it represents a realistic and real-

world application of workflow in health care. It also has specific and important characteristics, namely, it is

- context-aware;
- security-critical and
- time-critical.

workflow system in health care environment.

In $\mathcal{CAW}$, every bed (see Figure (7.1) is

- computerised with a touch display screen for
  - patients so they can simply touch the screen for entertainment purposes (watching television, DVD, play games and listening to radio) and
  - medical staff (nurses, doctors and consultants) so they can access medical data (x-rays, analysis, medication and/or historical health records) while working at the bed.
- aware of who is using it (i.e. the identity of the patient and staff). For example, the bed is aware of the nurse, the patient and the medicine tray;
- runs a context-aware Electronic Patient Record (*EPR*) client which communicate with a server which is reside in the hospital or near-by;

- according to the locations and access right of the *nurse*, *patient* and *medicine tray*, the bed can
  - automatically log in the nurse,
  - find the patient record,
  - display the medication records,
  - display the current prescribed medication which is in the *pill container*; and
  - automatically logged out the nurse/medical staff once they leaves, what is known as the *bed zone*[1] of the bed.
- the bed is also aware of all other environment changes that occur within its bed zone; for example, cleaning, inspection times, food time, etc.

This mechanism of "logging in" and "logging out" a user based on its proximity is called *proximity-based user authentication* [19, 20].

[1] Sometimes known as *active zone*.

Figure 7.1 – Context-Aware ward [19, 20]

As discussed in [20], context is more than location in a hospital setting. For example, the nurse documenting the medicine needs not be located close to the patient, even though this patient is definitely part of her work context. Moreover, (as we mentioned earlier) the patient's EPR[2] may be stored on a remote server but still provides valuable context information about the patient condition and treatment.

Here, we consider six contexts: these are

[2]Electronic Patient Record

1. bed,

2. patient,

3. nurse,

4. medicine tray,

5. pill container and

6. bed zone.

Each of these entities is represented by a context in $\mathcal{CS}-\mathcal{F}low$.

Initially, the bed is located in its *bed zone*, the *patient* is inside the *bed*, and the *nurse* and *medicine tray* containing the pill containers are outside the *bed zone* of the *bed*. This is modelled as follows. Let P_i be the activity describing the behaviour of the $context_i$ and let pat_i denotes the ith patient. The frame $\langle pat_i,\ \tilde{a}_i \rangle$ denotes the medications, a_i needed by patient pat_i.

Summary of the frames of these contexts are given in Table(7.1)

Table 7.1 – Contexts Frames

Frames	**Context**	**Frames**	**Context**
$\tilde{x}_n$	Nurse	$\tilde{x}_t$	Tray
$\tilde{W}$	Bed	$\tilde{W}_1$	Patient
$\tilde{W}_2$	Nurse Room	$\tilde{W}_3$	Medicine Room
$(pat\_1, \tilde{a}_1), \cdots (pat\_n, \tilde{a}_n)$	n *Drug containers*		

There is a number of channels which are used by the contexts and their activities. the following table, Table(7.2), summaries the name of these channels.

Table 7.2 – Channels: (a $\hookrightarrow$ b) means a to b

Channels	**Contexts**	**Channels**	**Contexts**
$chan_{n.t}$	Nurse $\hookrightarrow$ Tray	$chan_{t.n}$	Tray $\hookrightarrow$ Nurse
$chan_{pc.az}$	Pill container $\hookrightarrow$ Active zone	$chan_{pb}$	Patient $\hookrightarrow$ *bed*
$chan_{bp}$	bed $\hookrightarrow$ patient		

7.5 The $\mathcal{CAW}$ System

In this section we systematically develop the $\mathcal{CAW}$ system. The overall workflow of ward, $\mathcal{CAW}$, is given as follows:

(7.1)

$$\mathcal{CAW} \;\widehat{=}\; \left(\begin{array}{l} nurse\ \langle \tilde{x_n} \rangle\ :\ \{P_n\} \\ \|bed\langle \tilde{w} \rangle\ :\ \{P_b\} \\ \|patient\ \langle \tilde{w_1} \rangle\ :\ \{P_p\} \\ \|nurse-office\ \langle \tilde{w_2} \rangle\ :\ \{P_{n.o}\} \\ \|medicine-room\ \langle \tilde{w_3} \rangle\ :\ \{P_{m.r}\} \\ \| \\ tray\ \langle \tilde{x_t} \rangle\ :\ \{P_t \parallel Cont_1\ \langle pat_1,\ \tilde{a}_1 \rangle\ :\ \{P_1\} \\ \qquad \left(\begin{array}{ll} \| & Cont_2\ \langle pat_2,\ \tilde{a_2} \rangle\ :\ \{P_2\} \\ \| & \cdots\cdots\cdots \\ \| & Cont_k\ \langle pat_k,\ \tilde{a_k} \rangle\ :\ \{P_k\} \end{array} \right) \\ \qquad \} \end{array} \right)$$

where P_x is an activity specifying the behaviour of its host context.

7.5.1 The Nurse Context

The nurse

- can be in or out of the bed zone as often as needed in the course of her work activities;
- once inside the bed zone, the nurse can access the patient EPR using the touch screen embedded in the bed.
- brings the medicine tray as s/he enters and take it out as s/he leaves the bed zone;

- hands out the daily medication and handing out medication on demand (as described in Section (7.3) (page 226).

We note that

- "bringing medicine tray with him/her" requires a synchronisation between the activities representing the nurse and the activity modelling the medicine tray.

- the activity of handing out medication occurs at a regular intervals. The periodic activity given in Section (3.5) (page 102) can be utilised.

So the behaviour of the nurse can be modelled as follows:

$$
\begin{array}{ll}
(7.2) & P_n \mathrel{\widehat{=}} \texttt{chan}\ chan_{n.t}\ \texttt{in} \left\{ \begin{array}{l}
\texttt{while true} \\
\quad chan_{n.t}\ !\ \texttt{any}\ ; \\
\quad \texttt{to}(bed)\ ; \\
\qquad\qquad [epr(P_i) \parallel HandOutDrug(P_i)] \\
\quad chan_{n.t}\ !\ \texttt{any}\ ; \\
\quad \texttt{to}(nurse - office)
\end{array} \right\}
\end{array}
$$

where, given a period (T), deadline (D), and number of periods (N), such that $D \leq T$,

(7.3)

$$HandOutDrug \mathrel{\widehat{=}} \texttt{var}\ T,\ D,\ N,\ i\ \texttt{in} \left\{ \begin{array}{l} \texttt{while i} \leq \texttt{N} \\ \texttt{do} \\ \quad ([T]([D]GiveDrug) \parallel \texttt{delay}(T)\ )\ ; \\ \quad i = i + 1 \\ \texttt{od} \end{array} \right\}$$

7.5.2 The Tray Context

As we mentioned above, as the nurse goes in/out of the bed zone with the purpose of administering medication, it signals that fact to the tray context so as to follow the nurse (note that medicine in the tray context is stored in the pill containers context). So the tray context communicates with the nurse context to know when to move in and out the bed zone. This is modelled as follows and the explanation is similar to that of the nurse context.

(7.4)

$$P_t \mathrel{\widehat{=}} \texttt{chan}\ chan_{n.t}\ \texttt{in} \left\{ \begin{array}{l} \texttt{while true} \\ \quad chan_{n.t}\ ?\ \texttt{any}\ ; \\ \quad \texttt{to}(bed)\ ; \\ \quad\quad epr(P_i)\ ; \\ \quad\quad chan_{n.t}\ ?\ \texttt{any}\ ; \\ \quad\quad \texttt{to}(medicine - room) \end{array} \right\}$$

7.5.3 The Pill Container Context

The name of the patient will be identified to the pill container by lighting the patient's name. For example, if the pill container con_i, for some i such that $1 \leq i \leq k$, contains the medicine of the patient named *patient_001*, then the behaviour of the pill container is specified as follows:

(7.5)

$$P_i \mathrel{\widehat{=}} \texttt{chan}\ chan_{pc.c}\ \texttt{in} \left\{ \begin{array}{l} \texttt{while true} \\ \texttt{do} \\ \quad \texttt{somewhere}(p_t) \rightarrow chan_{pc.c}\ !\ 'patient\_001' \\ \texttt{od} \end{array} \right\}$$

To model this communication, P_t is modified as follows, to enable communication between the tray context and the pill container contexts:

(7.6)

$$P_t \mathrel{\widehat{=}} \texttt{chan}\ chan_{n.t},\ chan_{pc.c}\ \texttt{in} \left\{ \begin{array}{l} \texttt{while true} \\ \quad chan_{n.t}\ ?\ \texttt{any}\ ; \\ \quad chan_{pc.c}\ ?\ p_{name}\ ; \\ \quad \texttt{to}(bed)\ ; \\ \quad\quad epr(P_i)\ ; \\ \quad\quad chan_{n.t}\ ?\ \texttt{any}\ ; \\ \quad\quad \texttt{to}(medicine - room) \end{array} \right\}$$

The tray activity is now able to receive messages from the pill container context it contains. This name, in turns can be propagated/broadcast to other context using (see our Swift-Cabs Ltd in Section(3.5.5, on page 111)

$$chan_{nt} \,!\, p_{name} \parallel chan_{pc.c} \,!\, p_{name} \parallel \cdots$$

7.5.4 The Patient Context

The patient context selects an entertainment program by sending a request to the bed context as follows:

$$P_p \mathrel{\widehat{=}} \texttt{chan}\ chan_{P.b}, chan_{b.p}\ \texttt{in} \left\{ \begin{array}{l} \texttt{while true} \\ \quad chan_{p.b} \,!\, request \parallel \\ \quad chan_{b.p} \,?\, display \end{array} \right\} \tag{7.7}$$

The *freshness* of a request is guaranteed by the tight synchronisation model.

7.5.5 The Bed Context

One of the most important context-awareness properties of the bed is the ability of logging the nurse in when s/he enters the bed zone and logging her/him out when s/he leaves that zone, automatically. This is modelled as follows:

(7.8)

$$P_{\mathrm{b}} \mathrel{\widehat{=}} \left\{ \begin{array}{l} \texttt{while}(\texttt{true})\ \texttt{do}\ \{ \\ \quad \texttt{not somewhere}(\mathit{ActiveZone}) \ \rightarrow\ \mathit{logout}(n))\ ; \qquad (a) \\ \quad \Box \\ \quad \texttt{somewhere}(\mathit{ActiveZone})\ \texttt{and}\ \mathit{epr}(\mathit{req}, p(i)) \ \rightarrow\ \mathit{login}(n) \quad (b) \\ \qquad\qquad \} \end{array} \right\}$$

The context-guarded capability in Eq. (7.8)-*a* says that when the nurse enter the bed's bed zone, the activity '$\texttt{login}\langle \mathit{nurse} \rangle$' is taken to log the nurse in the EPR system. The nurse is logged out when she leaves the bed zone as specified in Eq. (7.8)-*b*.

7.6 Verification

In this section we concentrate on the *nurse* context and in particular the *HandOutDrug* activity described by the Equation(7.9) (page 239). For *every* six time units, *within* two time units, the nurse reads the patient's temperature, *TempC*, via a channel *read*, and then updates s/his temperature record in *Temp*[3]. This behaviour is *repeated* four times daily. This periodic activity is specified as follows:

(7.9)

$$HandOutDrug \mathrel{\widehat{=}} i := 1;\ [6 \times 4] \left(\begin{array}{l} \texttt{while i} \leq 4 \\ \texttt{do} \\ \quad [6]([2](read\,?\,TempC\ ;\ Temp := TempC) \parallel \texttt{delay}(6))\ ; \\ \quad i = i + 1 \\ \texttt{od} \end{array} \right)$$

The safety property that we need to proof is that the temperature is updated within two time units, every six time units, four time daily:

Theorem 7.6.1

(7.10) $$SafetyProp \mathrel{\widehat{=}} \texttt{every}(6, \texttt{within}(2, \mathcal{P}(t)))$$

Proof:

[3]The act of *taking temperature* can be considered as a special case of *HandOutDrug*.

$$1.\ \underline{(def, ?, \texttt{at})} \left(\begin{array}{l} read\ ?\ TempC\ ;\ \ Temp := TempC \end{array} \right) \models \bigvee_{\sigma \in [t_\alpha, t_\beta]} Temp\ \texttt{at}\ t_\beta = TempC$$

$$2.\ \underline{(1, Dur-1, \&-2)} \left(\begin{array}{l} \texttt{var}\ \ Temp,\ TempC,\ \ \texttt{chan}\ \ read\ \texttt{in} \\ \{ \\ read\ ?\ TempC\ ;\ \ Temp := TempC \\ \} \end{array} \right) \models \bigvee_{\sigma \in [t_\alpha, t_\beta]} Temp\ \texttt{at}\ t_\beta = TempC$$

$$3.\ \underline{(2, dur-3, \&-3a)} \left(\begin{array}{l} [2](\texttt{var}\ \ Temp,\ TempC, \\ \quad \texttt{chan}\ \ read \\ \texttt{in} \\ \{ \\ read\ ?\ TempC\ ;\ \ Temp := TempC \\ \}) \end{array} \right) \models \bigvee_{\sigma \in [t_\alpha, t_\beta]} Temp\ \texttt{at}\ t_\beta = TempC$$

$$4.\ \underline{(dur-4)} \left(\begin{array}{l} [2](\texttt{var}\ \ Temp,\ TempC,\ \ \texttt{chan}\ \ read\ \texttt{in} \\ \{ \\ read\ ?\ TempC\ ;\ \ Temp := TempC \\ \}) \end{array} \right) \models \texttt{duration} = 2$$

5. $\underline{(3,4,within-3a)}$ $\left(\begin{array}{l}[2](\texttt{var } Temp, TempC, \texttt{ chan } read \texttt{ in} \\ \{ \\ read\ ?\ TempC\ ;\ Temp := TempC \\ \})\end{array}\right) \models \texttt{within}(2,\mathcal{P}(t))$

6. $\underline{(dur-3)}\ \texttt{delay}[6] \models \texttt{durtion} \leq 6$

7. $\underline{(5,6,Dur-2)}$ $\left(\begin{array}{l}([2](\texttt{var } Temp, TempC, \texttt{ chan } read \texttt{ in} \\ \{ \\ read\ ?\ TempC\ ;\ Temp := TempC \\ \}))\ \|\ \texttt{delay}(6)\end{array}\right) \models \texttt{duration} \geq 6$

8. $\underline{(5,\|)}$ $\left(\begin{array}{l}([2](\texttt{var } Temp, TempC, \texttt{ chan } read \texttt{ in} \\ \{ \\ read\ ?\ TempC\ ;\ Temp := TempC \\ \}))\ \|\ \texttt{delay}(6)\end{array}\right) \models \texttt{within}(2,\mathcal{P}(t))$

9. $\underline{(8,Duration)}$ $\left(\begin{array}{l}[6](([2](\texttt{var } Temp, TempC, \texttt{ chan } read \texttt{ in} \\ \{ \\ read\ ?\ TempC\ ;\ Temp := TempC \\ \}))\ \|\ \texttt{delay}(6))\end{array}\right) \models \texttt{within}(2,\mathcal{P}(t))$

10. $\underline{(7, Duration)}$

$$\left(\begin{array}{l} [6](([2](\texttt{var}\ \mathit{Temp}, \mathit{TempC},\ \ \texttt{chan}\ \mathit{read}\ \texttt{in} \\ \{ \\ \mathit{read}\ ?\ \mathit{TempC}\ ;\ \ \mathit{Temp} := \mathit{TempC} \\ \}))\ \|\ \texttt{delay}(6)) \end{array}\right) \models \texttt{duration} \geq 6$$

11. $\underline{(10, Dur-4)}$

$$\left(\begin{array}{l} [6](([2](\texttt{var}\ \mathit{Temp}, \mathit{TempC},\ \ \texttt{chan}\ \mathit{read}\ \texttt{in} \\ \{ \\ \mathit{read}\ ?\ \mathit{TempC}\ ;\ \ \mathit{Temp} := \mathit{TempC} \\ \}))\ \|\ \texttt{delay}(6)) \end{array}\right) \models \texttt{duration} \leq 6$$

12. $\underline{(10, 11, \texttt{and})}$

$$\left(\begin{array}{l} [6](([2](\texttt{var}\ \mathit{Temp}, \mathit{TempC},\ \ \texttt{chan}\ \mathit{read}\ \texttt{in} \\ \{ \\ \mathit{read}\ ?\ \mathit{TempC}\ ;\ \ \mathit{Temp} := \mathit{TempC} \\ \}))\ \|\ \texttt{delay}(6)) \end{array}\right) \models \texttt{duration} = 6$$

13. $\underline{(9, 12, every-4)}$

$$\left(\begin{array}{l} [6 \times 4](\texttt{while}\ i \leq 4 \\ \texttt{do}\ [6] \\ ([2]\ (\mathit{read}\ ?\ \mathit{TempC}\ ;\ \ \mathit{Temp} := \mathit{TempC}) \\ \|\ \texttt{delay}(6))\ ; \\ i := i + 1 \\ \texttt{od} \end{array}\right) \models \texttt{every}(6, \texttt{within}(2, \mathcal{P}(t)))$$

■

The above demonstrates the systematic application of our timing proof system. The property is an important example of a large class of safety properties given the *time-critical* nature of many workflow systems. It is important to note that integrating our proof system with the equational laws provides a powerful tool for verification. A simple integration process can take the following form. Given a $\mathcal{CS}-\mathcal{F}low$ model, M and a property $Prop$

1. Apply algebraic laws to simplify the model and obtain an equivalent model M'. Such an application could be a *targeted* to a known model.

2. Apply the proof system on M'

The automation of the laws and the proof system are feasible and are left for future work.

7.7 Summary

The quest for a case-study with the purpose of evaluating $\mathcal{CS}-\mathcal{F}low$ is challenging. It is important however to elaborate on the precise meaning of "evaluation" of a process algebra such as the one presented here. We take the view that the choice of a case-study should be done to show key features of $\mathcal{CS}-\mathcal{F}low$; namely

- support concurrency;
- context and context awareness are first-class citizen;
- supports mobility as activities can move from one context to another;
- has the ability to express timing constrains: delay, deadlines, priority and schedulability;
- allows the expressibility of (*access control*) security policies without the need for an extra linguistic complexities; and
- enjoy sound formal semantics that allows us to animate design and compare various designs.

In this chapter, we have demonstrated the use of $\mathcal{CS}-\mathcal{F}low$ on a realistic case study from the health care domain, namely the Context Aware Ward ($\mathcal{CAW}$) system. Workflows within health care domain is inherently *security-critical* and

context-aware. Accessing patient's record must be done by appropriately authorised staff, the level of access varies according to the role a particular staff is holding (a Sargent or a consultant, for example will have full access to medical records whilst a nurse may not). In addition, as medical records take a variety of forms: textual (e.g., list of medication, description of diagnoses, etc.) and images (e.g., x-rays). These records need to be accessed (and some cases edited) on various devices with different operating systems and applications.

In addition, some activities within health care workflows may be time-critical, of a periodic nature and highly distributed across various hospital departments and between different hospitals and non-health care organizations (e.g., insurance companies, financial organisations, law-enforcement departments, etc.).

Chapter 8

CONCLUSION AND FUTURE WORK

Objectives

- To summaries and conclude our findings and put it all in context with other related work.
- Outline some future works to further advance the findings presented in the thesis.

8.1 Introduction

The evolution of businesses occurs as a result of the evolution of the environments within which they operate. The evolution of the environment can be technologically driven but can also be due to political, legislative and/or economic drivers: mergers of businesses, changes in governments policies, economic crises, etc. are few categories where environment may change. As a result, existing business processes often need to be re-engineered (to add or remove functionalities, to migrate systems to different platforms and/or languages, etc.) and to be optimised with the view to reduce costs, deliver timely services, and to enhance their competitive advantage in the market.

The philosophy of workflow technologies is to separate business policies/logics from the underlying business applications, hence enhancing the evolution of their processes and improving the re-engineering at the organisation level without the need to delve into the application details.

Workflow technologies have been applied in variety of domains including

1. office automation,
2. finance and banking,
3. health-care,

4. telecommunications,

5. military,

6. manufacturing and production.

We take the view that a workflow is a set of activities that achieve a common business objective. These activities are coordinated/orchestrated statically or dynamically and may be carried out by humans, application programs, or processing entities according to the organisational rules relevant to the process represented by the workflow. Activities within a workflow are usually related and dependent upon one another, which in turn are specified by a set of execution constraints such as concurrency, serialization, exclusion, alternation, compensation and so on.
An activity therefore

- has *a goal*,
- has *an input*,
- has *an output*,
- *performs in a particular order* – e.g. in sequence/parallel/alternate with others,
- is *associated with a particular context* – Contexts can be an organisation, a device, a service (e.g. web-based service) or a computational environment,

- has *timing constraints*,
- *uses resources/information*,
- *may affect more than one organisation unit* and
- *properly terminates* – in the same or in a different context [1].

For example, a workflow management system of a warehouse may have the following scenario: organises and controls the movement and storage of goods within a warehouse. All of these must be performed within a timing constraint. This is achieved through the definition and processing of complex transactions, including charging, insurance-provision, shipping, receiving, put away, picking and delivering of goods.

When executing workflow, a large number of different participants can be involved. These include, services and devices which may cross the boundaries of various organisations. There could be a multitude of heterogeneous devices ranging from resource-scarce sensor nodes, processor speed, capacity to powerful mainframes. The purpose of these workflows is to support human users in an unobtrusive way. This implies that, for example, users should not be constrained in their mobility. Users should always be able to interact with their workflows while at the same time the complexity of the network and its undesired properties like

[1] This is an important aspect of an activity as it impacts on and raises challenging issues regarding security and timing behaviours of an activity

e.g. intermittent communication links should be handled transparently. However, in networks that support mobility, devices and services may join or leave the network at all times and the quality of communication links may change over time. In order to support the accessibility of workflows in the face of intermittent communication links, to ensure the performance of the interactions in spite of varying link quality and to use the resources of the network efficiently, it is necessary to distribute a workflow in the network with respect to the current state of the environment (hence *context-variability* place a critical role in workflows). Moreover, due to the fact that changes in the environment happen spontaneously, we have to be able to distribute a workflow during runtime.

Consequently, this gives rise to some fundamental and challenging issues, such as *context variability*, *context-awareness* and *security*, which need to be considered at the specification, design and implementation stages of workflows. Therefore, it is important to be able to specify exact rules and constraints that prevent unauthorised participants from executing sensitive tasks and also to prevent tasks from accessing unauthorised services. Furthermore, workflows can hold and manipulate various data at different contexts and with different security requirements. Hence it is important to be able to *enforce* these requirements while the data is being accessed in a workflow instance. *Delegations* constraints over authorisations, *audit* and *integrity* provide additional security features. For example, medical sce-

narios will require that only authorised doctors are permitted to perform certain tasks and that only specific machines are used in those tasks. If a workflow execution *cannot guarantee* these requirements, then the workflow will be rejected.

Currently, there is no commonly accepted model for secure workflows or even a consensus on which features a workflow security model should support. Throughout the thesis, we have adopted a policy-based approach in which rules are specified compositionally as policies which can be analysed and continually verified at run-time. These are expressed using simple Boolean guards, over state variables and/or variables within context frames, within conditional or interrupt constructs. Most policy models that are available today, e.g. [6, 71], are of a static nature — it is difficult to express security requirements that are dependent on time or the occurrence of events, let alone represents contexts. Temporal aspects of access control are, for example, especially important in domains ranging from E-business to military domain where the value of tactical information, and therefore its protection requirements, are highly dependent on time (e.g. *time to start an activity*, *a period of time where an injection has to be induced* or period allowed to claim cost of a shipment) and events (e.g. *adversary action*, *coalition formation*, *new drugs came to market* or *more faster processor*).

Other work, e.g. [21, 23], has recognised the need for more expressive security policies to capture the temporal dimension of access control, however, these mod-

els lack *compositionality*. By compositionality we mean that the overall security policy can be composed out of smaller Boolean rules (policies). This is advantageous as each policy can capture specific requirements that can be validated and verified individually and are then composed to form the overall system policy.

Another important issue which has been highlighted by various authors, for example, [40, 134, 147] is that there is no workflow language available for the specification, modelling and designing context-aware workflow which takes into account security, timing and adaptability issues in a unified and coherent fashion. Although context adaptation can be considered as the exception to the normal workflow and hence may be handled through database triggers and *event-condition-action* rules, [33, 147], our central philosophy here is that context-awareness, timing and security issues are the norm in workflow systems which operate in real environments that are becoming more and more pervasive and complex.

8.2 Research Question

Once identified that workflow system and its management are intrinsically

- **context-aware**,
- and as they cross the boundaries of many organisations (contexts), they are **security-critical** and that security consideration is paramount,

- **highly distributed** and
- **time-critical**.

We were able to articulate the research question as follows:

How can we design, develop and build a context-aware, secure workflow system in an integrated fashion?

the quest to answer this question has led to various sub-questions which needed to be addressed. These were

1. What is the “nature" of *context* in workflow systems?
2. What are the various considerations or dimensions of “security" in the presence of mobility and context-awareness?
3. How do we “model" these system, with attributes such as being highly distributed, context-aware, time-dependent, allow mobility and being security-critical?
4. Policy-based approaches have been shown to be valuable tools, how can these policies be represented within context-awareness?
5. Being highly distributed, is there a more efficient mechanism for the design and development of context-aware, secure workflow systems that is amenable to analysis?

6. Is there a provision for animation?

This thesis has dealt with these sub-questions and provided a unique mechanism for answering the main research question.

- *1. What is the "nature" of context in workflow systems?* and

  2. What are the various considerations or dimensions of "security" for such systems?

  In Chapter (2), we have given a general overview of workflows and their importance in everyday businesses and enterprises. Often, workflows are conflated with business processes; In this thesis we have taken the view that both are identical unless we explicitly state the difference.

  There are two major concerns in current workflow system development. The first is **security** considerations and the second is **context awareness**.

  Modern workflow systems cross the boundaries of organisations, each has its own security requirements, policies and constraints. Even within one organisation, activities in a workflow systems may be executed, in one of its instances, within a platform but in another instance it may be executed or performed on a different platform with completely different environment. Indeed it may not even be automated.

  We have reviewed in some details Workflow Management Systems to-

gether with the associated Workflow Reference Model which is a de facto standard produced by the Workflow Management Coalition. We have also reviewed basic security requirements for workflow systems and it was clear that current Workflow Management Systems do not adequately deal with these requirements in the presence of contexts which change constantly.

In addition, within a highly dynamical environment, the notion of adaptability plays a central role in the design and implementation of workflow systems. There are various approaches to deal with adaptability of workflows which can be classified as run-time, automated and instance-based. However, none of them deal with adaptation at a secure context level.

- *3. How do we "model" these system, with these attributes, correctly?* and *4. Policy-based approaches have been shown to be valuable tools, how can these policies be context-aware?*

  In Chapter (3), we have given a detail description of the computational model for our Secure and context-aware workflows. It also presents our design language $\mathcal{CS}-\mathcal{F}low$. In our model, there are two dis-

tinct components: **Activity** and **context**. An **Activity** is the unit of computation – it is a piece of work that contributes toward the accomplishment of a given (functional and business) goal. An activity *starts, executes* and then *terminates*. Associated with an activity is one or more *context*. Contexts can be an organisation, a device, a service (e.g. web-based service) or a computational environment. An activity starts in one context and may terminates in a different context (hence its "association" with more than one context).

Each context is governed by a set of *security policies* which are continually changing due to either the occurrence of an event and/or the passage of time. In the model, activities may be composed concurrently to produce a new activity which terminates if and only if all of its components terminate.

It is also recognised that real-time is an important feature of an activity within workflow systems. From *deadlines*, and *worst case execution times* to *delays*, *priorities* and *interrupts*. We have carefully considered all these features and their associated specific constructs and have made the appropriate provisions. Furthermore, in dealing with real-time activities, our model has not made any of the usual simplified assumptions which are commonly used with real-time formalisms.

For example, the *maximal parallelism* hypothesis (i.e., the availability of an infinite number of resources), [129], and the instantaneous communication assumption [101, 132].

The model is supported by a design language, known as $\mathcal{CS}-\mathcal{F}low$, with which we are able to design and analyse workflows.

One powerful aspect of $\mathcal{CS}-\mathcal{F}low$ is in its Boolean guard which can be used to model context expressions as well as to describe security policies, (Section(3.5.7)). In addition to the usual propositional operators such as negation (`not`) and conjunction (`and`): $\texttt{somewhere}(\alpha)\cdot G$ is a *spatial* modal operator that hold if there exists (at least one) subcontext in which G holds. (We note that the dual of this guard, i.e. $\texttt{everywhere}(\alpha)\cdot G$ which holds if G hold everywhere within the context α can also be defined). There is also a provision for *mobility*, using the construct $\texttt{to}(\alpha)$, which upon execution, the current activity is moved to another context α.

We also gave a sound formal semantics for $\mathcal{CS}-\mathcal{F}low$ in a process algebraic style as well as algebraic characterisation of the language is also given (Chapter(4)). An *algebraic characterisation* of $\mathcal{CS}-\mathcal{F}low$

is also given which has a dual purpose: to provide an algebraic semantics of $\mathcal{CS}-\mathcal{F}low$ and also can be used to manipulate and analyse (at a textual level) $\mathcal{CS}-\mathcal{F}low$ models. Such type of semantics can be used as basis to develop equation theory that facilitates formal proofs.

- *5. Being highly concurrent, can we explore a more efficient mechanism for the design and development of context-aware, secure workflow systems?*

  We have presented the notion of **Communication-closed** layer (Chapter (5)). Such a notion can be seen as

  - design principle of secure workflow applications,
  - a layer-development methodology

  Such an approach is amenable for the introduction of

  - a programming construct which can offer a linguistic support for information flow security and
  - an efficient approach for both static and dynamic analyses for information security with an application such as that found in work flows.

- *6. Is there a provision for animation?*

  Towards this aim, we have developed *reduction rules* for $\mathcal{CS}-\mathcal{F}low$ in Chapter(6). Reduction rules describe how an activity can be executed

in a step-by-step fashion. Note that, the structural congruence, $\equiv$, (which we have given in Section(4.5)), together with the reduction relation $\rightarrow$ provide an operational semantics for $\mathcal{CS}-\mathcal{F}low$ and are used as the basis for validating (animating) by exploring various scenarios of designs/models of $\mathcal{CS}-\mathcal{F}low$ system.

The execution environment of `CCA` has been modified to include timing issues as well as priorities constructs. In this chapter we also gave the encoding procedure of $\mathcal{CS}-\mathcal{F}low$ to be suitable of execution together with few illustrative examples. The thesis ends with Chapter(7) which models a part of a *health care* system, known as $\mathcal{CAW}$.

8.3 Future Work

We list some of the many interesting outstanding issues which could be explored in the future:

- **Workflow Control Engine and distribution.** To run workflows, a **Workflow Control Engine** (WCE) is required. An architecture of a WCE which is capable of executing distributed workflows is needed. This requires detailed exploration of workflow *distribution*. One of the challenges is that when applications are modelled as workflows, normally, workflows are running on a workflow server. This implies that in order to interact with an

application users need access to the workflow server where the workflow representing the application is located. In a context-aware environment, the accessibility of a user applications should be as high as possible despite mobility. However, due to the restrictions of wireless communication, the communication link between a user device and a workflow server may break or become unusable (e.g. due to quality constraints). This results in restrained accessibility. Obviously there are several approaches which can be explored. These include: approaches with a centralized server, approaches with many servers and totally distributed approaches. Each has its own advantages and disadvantages.

Another challenge is related to reconfiguration. Due to the highly dynamics nature of the computing environment addressed here, it is difficult or even impossible to foresee the configuration of an environment before application startup or even during its execution. The WCE, therefore, has to enable various forms of adaptation/re-configuration, which alter the current state of the workflow distribution in the system. The purpose of these adaptations is to compute new workflow fragmentations and placements that are better suited regarding the current and projected behaviour of workflows and the expected context. Adaptations are triggered when the actual state of the context is violating the assumptions that led to the current distribution. If

the adaptation is executed after a violation, a reactive strategy is followed. However, if those deviations are anticipated before their actual occurrence, a proactive strategy is applied for adaptation. We want to reason over the current, past and future workflow behaviour to enable the following forms of adaptation (just to mention a few): context, human, run-time state and workflow re-planning.

- **Proof system and Model Checker.** We would like to develop a proof system to $\mathcal{CS}-\mathcal{F}low$ that allows us to proof various functional, timing, context and security properties of a workflow. For example to show that an activity will read sensor often enough so as (important) data are not missed.

  Further, we wish to develop a theory of *contextual equivalence* for proving properties about contexts and context-awareness. We can envisage, as a starting point, to adopt the style outlined in [102, 121] which is a standard way of testing that two activities are contextually equivalent if and only if they admit the same elementary observations whenever they are inserted inside any arbitrary enclosing activity. This however requires us to develop a hierarchical structure to context. Currently, context in $\mathcal{CS}-\mathcal{F}low$ have a "flat" (or linear) structure.

- **Property Language.** We recognise that there is a need for a formal language within which we may be able to express and prove desirable prop-

erties about workflows. Such a language should be expressive to cater for context, timing, mobility and performance properties as well as "functional" properties. We have already shown how specific timing properties can be expressed and reasoned about but there are more complex properties that we wish to show specially those which require a *continuous* time domain as oppose to discrete one. We wish to explore the hybrid nature of workflow systems which may help in expressing various performance properties, for example:

- *"within t time unite it is likely that the temperature will rise above the acceptable threshold",*
- *" The employees in zone X should be trusted to a certain level",*
- *"the probability to satisfactorily complete a particular activity is 75%",*

  etc.

For this, we envisage using an extended version of a Temporal Logic (e.g. Interval Temporal Logic(ITL) [34], Probabilistic Computation Tree logic (PCTL) [84] or Temporal Logic of Actions (TLA) [85]). This will require mechanisms to integrate both formalisms, for example through the unification of their semantic domains. A traditional way of doing so is by deriving the formal semantics of $\mathcal{CS}-\mathcal{F}low$ in the chosen logic, which render $\mathcal{CS}-\mathcal{F}low$ to be *Wide-Spectrum* formalism and framework. We envisage

that PCTL to be more appropriate as it has provision to express performance properties.

Appendix A

$\mathcal{CS}-\mathcal{F}low$: Algebraic Laws

A.1 Declaration

$$(Decl\ -1) \quad \texttt{var}\ \tilde{x}\ \texttt{in}\ \{P\} \equiv P\ \ (if\ x\ is\ not\ in\ P)$$

$$(Decl\ -2) \quad \texttt{chan}\ \tilde{x}\ \texttt{in}\ \{P\} \equiv P\ \ (if\ x\ is\ not\ in\ P)$$

$$(Decl\ -3) \quad \texttt{var}\ \tilde{x}\ \texttt{in}\ \{\texttt{var}\ \tilde{y}\ \texttt{in}\ P\} \equiv \texttt{var}\ \tilde{y}\ \texttt{in}\ \{\texttt{var}\ \tilde{x}\ \texttt{in}\ P\}$$
$$\equiv \texttt{var}\ \tilde{x},\ \tilde{y}\ \texttt{in}\ \{\ P\}$$

A.2 Delay

$$(\texttt{delay}\ -1) \quad \texttt{delay}\ (t_1)\ ;\ \texttt{delay}\ (t_2) \equiv \texttt{delay}\ (t_1 + t_2)$$

$$(\texttt{delay}\ -2) \quad \texttt{delay}\ (0)\ ;\ P \equiv P$$

A.3 Deadline

$$(Deadline\ -0) \quad [0]\,P \equiv \texttt{skip}$$

$$(Deadline\ -1) \quad [t_P]\,P\ ;\ [t_Q]\,Q \equiv [t_P,\,t_Q]\,(P\ ;\ Q)$$

$$(Deadline\ -2) \quad [t_P]\,P \parallel [t_Q]\,Q \sqsubseteq [\max\,(t_P,\,t_Q)](P \parallel Q)$$

A.4 Sequential

$$(;\ -1) \quad P\ ;\ (Q\ ;\ R) \equiv (P\ ;\ Q)\ ;\ R$$

$$(;\ -2) \quad Q\ ;\ \texttt{skip} \equiv Q$$

$$\equiv \texttt{skip}\ ;\ Q$$

A.5 Alternative

$$(Alt\ -1) \quad (G \to P \,\square\, G \to P) \equiv G \to P$$

$$(Alt\ -2) \quad (G \to P \,\square\, \texttt{not}\,G \to P) \equiv P$$

$$(Alt\ -3) \quad (G \to P \,\square\, \texttt{not}\,G \to Q) \equiv \texttt{not}\,G \to Q \,\square\, G \to P$$

$$(Alt\ -4) \quad (c \to (b \to P \,\square\, \texttt{not}\,b \to Q) \,\square\, \texttt{not}\,c \to R) \equiv$$

$$(b\ \texttt{and}\ c) \to P \,\square\, \texttt{not}\,(b\ \texttt{and}\ c) \to (c \to Q \,\square\, \texttt{not}\,c \to R)$$

$$(Alt\ -5) \quad b \ \rightarrow\ P \,\square\, \texttt{not}\ b \ \rightarrow\ (c \ \rightarrow\ P \,\square\, \texttt{not}\ c \ \rightarrow\ Q) \ \equiv$$

$$b\ \texttt{or}\ c \ \rightarrow\ P \,\square\, \texttt{not}\ (b\ \texttt{or}\ c) \ \rightarrow\ Q$$

$$(Alt\ -6) \quad b \ \rightarrow\ P \,\square\, \texttt{not}\ b \ \rightarrow\ (b \ \rightarrow\ Q \,\square\, \texttt{not}\ b \ \rightarrow\ R) \ \equiv$$

$$b \ \rightarrow\ P \,\square\, \texttt{not}\ b \ \rightarrow\ R$$

$$(Alt\ -7) \quad G_1 \ \rightarrow\ P \,\square\, G_2 \ \rightarrow\ Q \ \equiv\ G_2 \ \rightarrow\ Q \,\square\, G_1 \ \rightarrow\ P$$

$$(Alt\ -8) \quad (G_1 \rightarrow R \,\square\, G_2 \rightarrow S)\ ;\ P \ \equiv\ G_1 \rightarrow (R\ ;\ P) \,\square\, G_2 \rightarrow (S\ ;\ P)$$

$$(Alt\ -9) \quad \texttt{true} \rightarrow P \,\square\, \texttt{false} \rightarrow Q \ \equiv\ P$$

$$(Alt\ -10) \quad \texttt{false} \rightarrow P \,\square\, \texttt{true} \rightarrow Q \ \equiv\ Q$$

$$(Alt\ -11) \quad (G_1 \ \rightarrow\ P_1 \,\square\, G_2 \ \rightarrow\ P_2) \ \sqsubseteq\ P_2\ (if\ G_2 \equiv \texttt{true})$$

$$(Alt\ -12) \quad G_1 \rightarrow R \,\square\, G_2 \rightarrow S \ \equiv\ (G_1\ \texttt{and}\ R)\texttt{or}(G_2\texttt{and}\ S)$$

$$(Alt\ -13) \quad (G_1 \ \rightarrow R \,\square\, G_2 \ \rightarrow S) \,\square\, G_3 \ \rightarrow\ P \ \equiv\ (G_1 \ \rightarrow R) \,\square\, (G_2 \ \rightarrow S \,\square\, G_3 \ \rightarrow\ P)$$

$$(Alt\ -14) \quad (G\ \texttt{and}\ (G_1 \ \rightarrow\ P \,\square\, G_2 \ \rightarrow\ Q) \ \equiv\ G\ \texttt{and}\ G_1 \ \rightarrow\ P \,\square\, G\ \texttt{and}\ G_2 \ \rightarrow\ Q)$$

$$(Alt\ -15) \quad a \ \rightarrow\ P \,\square\, (b \ \rightarrow Q \,\square\, c \ \rightarrow\ R) \ \equiv\ (a \ \rightarrow\ P \,\square\, b \ \rightarrow Q) \,\square\, (a \ \rightarrow\ P \,\square\, c \ \rightarrow\ R)$$

A.6 Interrupt

$$(Interrupt\ -1) \quad (P \triangleright_t^G Q) \parallel (S \triangleright_t^G T) \equiv (P \parallel S) \triangleright_t^G (Q \parallel T)$$

$$(Interrupt\ -2) \quad P \triangleright_{t+1}^G Q \equiv (P \triangleright_t^G Q) \triangleright_1^G Q$$

$$(Interrupt\ -3) \quad P \triangleright_t^G (Q \triangleright_0^G R) \equiv P \triangleright_t^G Q \quad (if\ G\ has\ become\ false\ during\ t)$$

$$(Interrupt\ -4) \quad (P \triangleright_t^{\texttt{true}} Q)\ ;\ R \equiv Q\ ;\ R$$

A.7 Parallel

$$(\parallel -1) \quad P \parallel Q \equiv Q \parallel P$$

$$(\parallel -2) \quad P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R$$

$$(\parallel -3) \quad P \parallel \texttt{skip} \equiv P \equiv \texttt{skip} \parallel P$$

$$(\parallel -4) \quad \langle\,\rangle \parallel \langle\,\rangle \equiv \texttt{skip}$$

$$(\parallel -5) \quad (Chan!v)\ ;\ P \parallel (Chan?x)\ ;\ Q \equiv P \parallel x := v\ ;\ Q$$

$$(\parallel -6) \quad P \parallel (G_1 \to R \,\Box\, G_2 \to S) \equiv G_1 \to (R \parallel P) \,\Box\, G_2 \to (S \parallel P)$$

A.8 Assignment

$(asgn-1) \quad <>:=<> \;\equiv\; \texttt{skip}$

A.9 Somewhere

$(somewhere-1) \quad \texttt{somewhere}(\alpha) \cdot G\,\{P\} \parallel \texttt{somewhere}(\alpha) \cdot G\,\{Q\}$

$\equiv$

$\texttt{somewhere}(\alpha) \cdot G\,\{P \parallel Q\}$

$(somewhere-2) \quad \texttt{somewhere}(\alpha) \cdot G_1\,\{P\} \parallel \texttt{somewhere}(\alpha) \cdot G_2\,\{P\}$

$\equiv$

$\texttt{somewhere}(\alpha) \cdot (G_1 \text{ and } G_2)\,\{P\}$

$(somewhere-3) \quad \texttt{somewhere}(\alpha) \cdot \texttt{true}\,\{\texttt{skip}\} \equiv \texttt{skip}$

$(somewhere-4) \quad \texttt{somewhere}(\alpha) \cdot \texttt{true}\,\{\texttt{abort}\} \equiv \texttt{abort}$

(Note that $(Somewhere3\& - 4)$ can be generalised to:

$\texttt{somewhere}(\alpha) \cdot \texttt{true}\,\{P\} \equiv P)$

Note that (somewhere-1) is a special case of (somewhere-2) and that (somewhere-3) and (somewhere-4) are special cases of: $\texttt{somewhere}(\alpha).\texttt{true}P \equiv P$

A.10 abort

$(abort-1)$ $\texttt{abort} \equiv \texttt{while true do skip od}$

$(abort-2)$ $\texttt{abort ; } P \equiv \texttt{abort}$

$(abort-3)$ $\texttt{abort} \parallel P \equiv \texttt{abort}$

A.11 Loop

$(Loop-1)$ $P \equiv Q \Rightarrow \texttt{while}\, G\ \{P\} \equiv \texttt{while}\, G\ \{Q\}$

$(Loop-2)$ $\texttt{while}\, G\ \{P\} \equiv G \rightarrow P\ ;\ \texttt{while}\, G\ \{P\}$

Appendix B

$\mathcal{CS}-\mathcal{F}low$: Structure Congruence

Table B.1 – Structural congruence for activities

(S1)	$P \equiv P$
(S2)	$P \equiv Q \quad \Rightarrow \quad Q \equiv P$
(S3)	$P \equiv Q, Q \equiv R \quad \Rightarrow \quad P \equiv R$
(S4)	$P \equiv Q \quad \Rightarrow \quad P \parallel R \equiv Q \parallel R$
(S5)	$P \equiv Q \quad \Rightarrow \quad \alpha \langle \tilde{x} \rangle : \{P\} \equiv \alpha \langle \tilde{x} \rangle : \{Q\}$

Appendix C

$\mathcal{CS}-\mathcal{F}low$: Reduction Rules

Table C.1 – Reduction Rules-1

$P \to P'$	$\Rightarrow$	$\texttt{var}\ \tilde{x}\ \texttt{in}\ \{P\} \to \texttt{var}\ \tilde{x}\ \texttt{in}\ \{P'\}$	(Reduction Var)
$P \to P'$	$\Rightarrow$	$\texttt{chan}\ \tilde{x}\ \texttt{in}\ \{P\} \to \texttt{chan}\ \tilde{x}\ \texttt{in}\ \{P'\}$	(Reduction Chan)
$P \to P'$	$\Rightarrow$	$\alpha\ \langle\tilde{x}\rangle : \{P\} \to \alpha\ \langle\tilde{x}\rangle : \{P'\}$	(Reduction Contxt)
$P \to P'$	$\Rightarrow$	$\mathcal{C}(P) \to \mathcal{C}(P')$	(Reduction Contxt)
$P \to P'$	$\Rightarrow$	$P \parallel Q \to P' \parallel Q$	(Reduction Par)
$P \equiv Q,\ Q \to Q',\ Q' \equiv P'$	$\Rightarrow$	$P \to P'$	(Reduction $\equiv$)

Table C.2 – Reduction Rules - 2

$(Chan\ ?\ \tilde{y})\ ;\ P \parallel (Chan\ !\ \tilde{z})\ ;\ Q$

$\rightarrow\ P\{\tilde{y} \leftarrow \tilde{z}\} \parallel Q$ (Reduction Com-1)

$\alpha : \{(Chan\ ?\ \tilde{y})\ ;\ P \parallel Q\} \parallel \beta : \{(Chan\ !\ \tilde{z})\ ;\ R \parallel S\}$

$\rightarrow\ \alpha : \{P(\tilde{y} \leftarrow \tilde{z}) \parallel Q\} \parallel \beta : \{R \parallel S\}$ (Reduction Com-2)

$\alpha : (Chan\ ?\ \tilde{y}\ ;\ P \parallel Q) \parallel \beta : (\alpha : (Chan\ !\ \tilde{z}\ ;\ R \parallel S)$

$\rightarrow\ \alpha : (P(\tilde{y}\ \leftarrow\ \tilde{z})) \parallel \beta : (R \parallel S)$ (Reduction Com-3)

$\alpha : (\beta : (Chan\ ?\ \tilde{y}\ ;\ P \parallel Q)) \parallel \beta : (Chan\ !\ \tilde{z}\ ;\ R \parallel\ )$

$\rightarrow\ \alpha : (P(\tilde{y}\ \leftarrow\ \tilde{z}) \parallel Q) \parallel \beta(R \parallel S)$ (Reduction Com-4)

$\alpha : (\beta : (Chan\ ?\ \tilde{y}\ ;\ P \parallel Q)) \parallel \beta : (\alpha : ((Chan\ !\ \tilde{z}\ ;\ R \parallel\ ))$

$\rightarrow\ \alpha : (P(\tilde{y}\ \leftarrow\ \tilde{z}) \parallel Q) \parallel \beta(R \parallel S)$ (Reduction Com-5)

$\beta :\ \{\mathtt{to}(\alpha)\ ;\ P \parallel Q\}\ \parallel\ \alpha :\ \{R\}$

$\rightarrow\ \alpha :\ \{\beta : \{P \parallel Q\}\ R\}$ (Reduction Mob)

Appendix D

Proof Rules for Timing Properties

duration ($\mathcal{D}$), *within* ($\mathcal{W}$), *after* ($\mathcal{A}$), *between* ($\mathcal{B}$) and *every* ($\mathcal{E}$).

D.1 duration – $\mathcal{D}$

Let $primitive \in \{!, ?, x := v, \texttt{skip}\}$. We have

$$(Dur-0)\ \frac{}{primitive \models \mathcal{D} \geq 1}$$

$$(Dur-1)\ \frac{P \models \mathcal{D} \geq n, \quad Q \models \mathcal{D} \geq m}{P\ ;\ Q \models \mathcal{D} \geq (n+m)}$$

$$(Dur-2)\ \frac{\begin{array}{l} P \models \mathcal{D} \geq n, \\ Q \models \mathcal{D} \geq m \end{array}}{P \parallel Q \models \mathcal{D} \geq \max(n,m)}$$

$$(Dur-3)\ \frac{P \models \mathcal{D} \geq n}{[S]\,P \models \mathcal{D} \geq n}$$

$$(Dur-3a)\ \frac{P \models \Phi}{[S]\,P \models \Phi}$$

$$(Dur-4)\ \frac{}{(Dur-4)\ [S]\,P \models \mathcal{D} \in S}$$

$$(Dur-5)\ \frac{\begin{array}{l} P \models \mathcal{D} \geq n, \\ G \end{array}}{(\texttt{while}\ G\ \texttt{do}\ P\ \texttt{od}) \models \mathcal{D} \geq n}$$

$$(Dur-6)\ \frac{\begin{array}{l} P \models \mathcal{D} \geq n, \\ Q \models \mathcal{D} \geq m, \\ G_1\ \texttt{at}\ t_\alpha \vee G_2\ \texttt{at}\ t_\alpha \end{array}}{(G_1 \rightarrow P \,\square\, G_2 \rightarrow Q) \models \mathcal{D} \geq \min(n,m)}$$

$$(Dur-7)\ \frac{P \models \mathcal{D} \geq n}{(P \rhd_t^{\texttt{false}} Q) \models \mathcal{D} \geq (n+t)}$$

$$(Dur-8)\ \frac{}{\texttt{delay}(n)\ \models\ \mathcal{D}\ \geq\ n}$$

D.2 within – $\mathcal{W}$

$$(within-1)\ \frac{P \models\ \mathcal{W}\,(n, \mathcal{P}(t))}{P\ ;\ Q\ \models\ \mathcal{W}\,(n, \mathcal{P}(t))}$$

$$(within-2)\ \frac{P \models\ \mathcal{W}\,(n, \mathcal{P}(t))}{[S]P\ \models\ \mathcal{W}\,(n, \mathcal{P}(t))}$$

$$(within-3)\ \frac{\begin{array}{l} P\ \models\ \mathcal{W}\,(n, \mathcal{P}(t)), \\ P\ \models\ \mathcal{D}\ =\ (n+m) \end{array}}{P\ \models\ \mathcal{W}\,(n+m,\ \mathcal{P}(t))}$$

$$(within-3a)\ \frac{\begin{array}{l} P\ \models\ \bigvee_{t\,\in\,[t_\alpha, t_\beta]}\ \mathcal{P}(t) \\ P\ \models\ \mathcal{D}\ \leq\ n \end{array}}{P\ \models\ \mathcal{W}\,(n, \mathcal{P}(t))}$$

$$(within-4)\ \frac{\begin{array}{l} P\ \models\ \mathcal{D}\ \leq\ n, \\ P\ \models\ \bigvee_{\sigma\,\in\,[t_\alpha, t_\beta]}\ \mathcal{P}(t) \end{array}}{P\ \models\ \mathcal{W}\,(n, \mathcal{P}(t))}$$

$$(within-5)\ \frac{P\ \models\ \mathcal{D}\ \leq\ n}{P \,\|Q\ \models\ \mathcal{W}\,(n, \mathcal{P}(t))}$$

D.3 after – $\mathcal{A}$

$$(after-1)\ \frac{P \models \mathcal{A}\,(n,\mathcal{P}(t))}{P\ ;\ Q \models \mathcal{A}\,(n,\mathcal{P}(t))}$$

$$(after-2)\ \frac{P \models \mathcal{A}\,(n,\mathcal{P}(t))}{P \parallel Q \models \mathcal{A}\,(n,\mathcal{P}(t))}$$

$$(after-3)\ \frac{\begin{array}{l} Q \models \mathcal{A}\,(n,\mathcal{P}(t)), \\ P \models \mathcal{D} \geq m \end{array}}{P\ ;\ Q \models \mathcal{A}\,(n+m,\mathcal{P}(t))}$$

D.4 between – $\mathcal{B}$

$$(between-1)\ \frac{A \models \mathcal{B}\ (P(t),n,Q(t))}{A \parallel B \models \mathcal{B}\ (P(t),n,Q(t))}$$

$$(between-2)\ \frac{A \models \mathcal{B}\ (P(t),n,Q(t))}{A\ ;\ B \models \mathcal{B}\ (P(t),n,Q(t))}$$

$$(between-3)\ \frac{\begin{array}{l} A \models \mathcal{B}\ (P(t),n,Q(t)), \\ B \models \mathcal{B}\ (P(t),m,Q(t)) \\ m \leq n \end{array}}{A\ ;\ B \models \mathcal{B}\ (P(t),n,Q(t))}$$

$$(between-4)\quad \frac{\begin{array}{l} A \models \mathcal{B}\ (P(t), l, Q(t)), \\ B \models \mathcal{B}\ (P(t), l, Q(t)), \\ A \models \mathcal{D} = n, \\ B \models \mathcal{D} = n \end{array}}{A \parallel B \models \mathcal{B}\ (P(t), l, Q(t))}$$

D.5 every – $\mathcal{E}$

$$(every-1)\quad \frac{\begin{array}{l} m \geq n, \\ P \models \mathcal{A}\ (n, \mathcal{P}(t)), \\ P \models \mathcal{D} = m \end{array}}{(\texttt{while}\ G\ \texttt{do}\ P\ \texttt{od}) \models \mathcal{E}\ (m,\ \mathcal{A}\ (n, \mathcal{P}(t)))}$$

$$(every-2)\quad \frac{\begin{array}{l} P \models \mathcal{E}\ (n, \mathcal{P}(t)), \\ Q \models \mathcal{E}\ (m, Q(t)) \end{array}}{(P \parallel Q) \models \mathcal{E}\ (n, \mathcal{P}(t)) \wedge \mathcal{E}\ (m, Q(t))}$$

$$(every-3)\quad \frac{\begin{array}{l} P \models \mathcal{D} = l, \\ \exists m \bullet (m.n = l), \\ P \models \mathcal{E}\ (n, \mathcal{P}(t)), \\ Q \models \mathcal{E}\ (n, \mathcal{P}(t)) \end{array}}{(P\ ;\ Q) \models \mathcal{E}\ (n, \mathcal{P}(t))}$$

$$(every-4)\quad \frac{P \models \mathcal{W}(l, \mathcal{P}(t)),\quad P \models \mathcal{D} = m}{([m.n]\ \texttt{while}\ G = n\ \texttt{do}\ P\ \texttt{od}) \models \mathcal{E}(m, \mathcal{W}(l, \mathcal{P}(t)))}$$

D.6 General Rules

$$(Weaken)\quad \frac{P \models \Phi \qquad \Phi \Rightarrow \Psi}{\Psi \models P}$$

$$(and)\quad \frac{P \models \Phi \qquad P \models \Psi}{P \models \Phi \wedge \Psi}$$

Bibliography

[1] Ws-diamond

http://wsdiamond.di.unito.it/, 2003.

[2] Oasis: ws-security,

http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss#technical,

2005.

[3] Oasis: ws-security,

http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security#technical,

2006.

[4] Ukcrc grand challenges for computing research ubiquitous computing: Science and design. http://www-dse.doc.ic.ac.uk/Projects/UbiNet/GC/, 2006.

[5] Worklets: A service-oriented implementation of dynamic flexibility in work-flows. In *OTM Conferences*, volume 1, pages 291–308, 2006.

[6] Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A Calculus for Access Control in Distributed Systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, September 1993.

[7] E. E. Almeida, J. E. Luntz, and D. M. Tilbury. Event-condition-action systems for reconfigurable logic control. *IEEE Transactions on Knowledge and Data Engineering*, 4((2)):167 – 181, (2007).

[8] H. Alotaibi and H. Zedan. The design and analysis of context-aware, secure workflow systems. In *Proceedings of the 40th Informatics and Mathematics*, 2010.

[9] H. Alotaibi and H. Zedan. Context-aware and secure workflow systems. *To be submitted to: Journal of Software and Information Technology*, 2012.

[10] Hind Alotaibi and Hussein Zedan. An agent-based approach for policy enforcement in secure workflow systems. In *Intelligent Systems Design and Applications (ISDA)*, pages 348–355. IEEE, 2010.

[11] Hind Alotaibi and Hussein Zedan. Runtime verification of safety properties in multi-agents systems. In *Intelligent Systems Design and Applications (ISDA)*, pages 356–362. IEEE, 2010.

[12] L. Ardissono, R. Furnari, A. Goy, G. Petrone, and M. Segnan. An event-based model for the management of choreographed services. In *Proc. of 9th*

International Conference on Electronic Commerce and Web Technologies (EC-Web08, DEXA), 2008.

[13] L. Ardissono, R. Furnari, A. Goy, G. Petrone, and M. Segnan. A soa-based model supporting adaptive web-based applications. In *Proc. of 3rd IEEE Conference on Internet and Web Applications and Services (ICIW 2008)*, pages 708–713, 2008.

[14] Huang W.-K. Atluri, V. and E. Bertino. An execution model for multilevel secure workflows. In *Proceedings of the 11th IFIP Working Conference on Database Security*, pages 151–165, 1997.

[15] V. Atluri and W.-K Huang. An extended petri net model for supporting workflows in a multilevel secure environment. In *Proceedings of the 10th IFIP WG 11.3 Working conference on Database Security*, pages 240–258, 1996.

[16] R.C. Backhouse and P. Hoogendijk. Elements of the relational theory of data types. *Lecture Notes in Computer Science*, 755, 1993.

[17] J. Bae, H. Bae, S.-H. Kang, and Y. Kim. Automatic control of work ow processes using eca rules. *IEEE Transactions on Knowledge and Data Engineering*, 16((8)):1010–1023, (2004).

[18] F. Baiardi, L. Ricci, A. Tomasi, and M. Vannechi. Structuring process for a cooperative approch to fault-tolerant distributed software. In *Proc. of 4th IEEE Symposium on reliability in Distributed Software and Database Systems*, pages 218–230, 1984.

[19] J. Bardram. Hospitals of the future–ubiquitous computing support for medical work. In *in: Hospitals Workshop Ubihealth 2003*, 2003.

[20] J. Bardram. Aplications of context-aware computing in hospital work–examples and design principles. In *in: Proceedings of ACM Symposium on Applied Computing*, pages 1574–1579, 2004.

[21] Steve Barker and Peter J. Stuckey. Flexible Access Control Specification with Constraint Logic Programming. *ACM Transactions on Information and System Security*, 6(4):501–546, November 2003.

[22] T. Beer, J. Rasinger, W. Hopken, M. Fuchs, and H. Werthner. Exploiting e-c-a rules for designing and processing context-aware push messages. *Lecture Notes in Computer Science*, 4824:199–206, (2007).

[23] Elisa Bertino, Piero Andrea Bonatti, and Elena Ferrari. TRBAC: A temporal role-based access control model. *ACM Transactions on Information and System Security*, 4(3):191–233, 2001.

[24] R. Bhatti, J.B.D. Joshi, E. Bertino, and A. Ghafoor. Access control in dynamic xml-based web-services with xrbac. In *Proceedings of Intern. Conf. of Web Services*, 2003.

[25] L. Birkedal, S. Debois, E. Elsborg, T. Hildebr, and H. Niss. Bigraphical models of context-aware systems. In *in: IT University of Copenhagen, Springer-Verlag*, pages 187–201, 2006.

[26] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. 1999.

[27] K. Boukadi, C. Ghadira, S. Chaari, L. Vincent, and L. Batainah. Cwsc4ec: how to employ context, web service and community in enterprise collaboration. In *Procceding of the 8th Intern. Conferefnce on New Technologies in Distributed Systems*, 2001.

[28] A. Bouzeghoub, K. N. Do, and L. K. Wives. Situation-aware adaptive recommendation to assist mobile users in a campus environments. In *in: International Conference on Advanced Information Networking and Applications, IEEE Computer Society*, pages 503–508, 2009.

[29] A. Brogi and R. Popescu. Automated generation of bpel adapters. In *International Conference on Service Oriented Computing*, 2006.

[30] D. Bucur and M. Nielsen. Secure data flow in a calculus for context awareness. *Lecture Notes in Computer Science*, 5065:439–456, 2008.

[31] He Jifeng C.A.R. Haore and A. Sampaio. Normal form pproach to compiler design. *Acta Informatica*, 2:701 – 739, 1993.

[32] L. Cardelli and A. Gordon. Mobile ambients. *Theoretical Computer Science*, 240:177–213, (2000).

[33] F. Casati, S. Ceri, S. Paraboschi, and G. Pozzi. Specification and implementation of exceptions in workflow management systems. *ACM Transactions on database systems*, 24(3), 1999.

[34] Antonio Cau, Ben Moszkowski, and Hussein Zedan. Interval temporal logic www site. `http://www.cse.dmu.ac.uk/STRL/ITL/`.

[35] Zhou Chachen, C.A.R. Hoare, and A.P. Ravn. A calculus of duration. *Information Processing Letters*, 40(5):269–276, 1992.

[36] G. Chafle, K. Dasgupta, A. Kumar, S. Mittal, and B. Srivastava. Adaptation in web service composition and execution. In *International Conference on Web Services - ICWS*, pages 549–557, 2006.

[37] Dipanjan Chakraborty and Hui Lei. Pervasive enablement of business processes. In *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*.

[38] G. Chen. A survey of context-aware mobile computing research, 2000. Dartmouth College, Dartmouth.

[39] H. Chen, T. Finin, and A. Joshi. An ontology for context-aware pervasive computing environments. *Knowl. Eng. Rev.*, 18 (3):197–207, (2003).

[40] Y. Cho, J. Choi, and J. Choi. A context-aware workflow system for a smart home. In *Proc. of the International Conference on Convergence Information Technology*, pages 95 – 100, 2007. DOI 10.1109/ICCIT.2007.263.

[41] J. Davis. *Specification and Proof in Real-Time Systems*. PhD thesis, Oxford University, 1991.

[42] A. K. Day. Understanding and using contexts. *Personal and Ubiquitous Computing*, 5(1):4–7, 2001.

[43] E.W. Dijkstra. *A discipline of programming*. Prentice–Hall, 1972.

[44] Tzilla Elrad. The use of communication-closed layers to support imprecise scheduling for distributed, real-time programs, 1991.

[45] Tzilla Elrad and Nissim Francez. Decomposition of distributed programs into communication-closed layers. *Science of Computer Programming*, 2:155–173, 1982.

[46] Maarten Fokkinga, Mannes Poel, and Job Zwiers. Modular completeness for communication closed layers. In E. Best, editor, *CONCUR'93*, Lecture Notes in Computer Science 715, pages 50–65, 1993. Imported from EWI/DB PMS [db-utwente:inpr:0000003411].

[47] Hussein Zedan Francois Siewe and Antonio Cau. The calculus of context-aware ambients. *Journal of Computer and System Science*, 2010. 10.1016/j.jcss.2010.02.003.

[48] D. Tosi G. Denaro, M. Pezze and D. Schilling. Towards self-adaptive service-oriented architectures. In *TAV-WEB'06: Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications*, pages 10–16, 2006.

[49] Michael George and Jon Pyke. Dynamic process orchestration. In *White paper, Stallware PLC*, 2003.

[50] M. Grossmann, M. Baur, N. Honle, and U.-P. Kappeler. Efficiently managing context information for large-scale scenarions. In *Proceedings of 3ed Intern. Conference on Pervasive Computing and Communications*, 2005.

[51] C.A. Gunter. *Semantics of Programming Languages: Structues and Techniques*. MIT Press, 1992.

[52] D. Harel. Statecharts: A visual formulation for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[53] D. Harel and A. Naamad. The statemate semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, 1996.

[54] D. Harel and A. Pnueli. On the development of reactive systems. *Lecture Notes in Computer Science*, 13:447–498, 1985.

[55] J. Harney and P. Doshi. Adaptive web processes using value of changed information. In *International Conference on Service-Oriented Computing (ICSOC)*.

[56] John Harney and Prashant Doshi. Speeding up adaptation of web service compositions using expiration times. In *Proceedings of WWW'07*, 2007.

[57] E.C.R. Hehner. Predictive programming. *Communications of the ACM*, 27(2):134–151, 1984.

[58] E.C.R. Hehner and C.A.R. Hoare. A more complete model of communicating processes. *Theoretical Computer Science*, 26:105 – 120, 1983.

[59] M.C. Hennessy. *Algebraic theory of processes*. MIT, 1988.

[60] M.C. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, pages 137 – 161, 1985.

[61] M. Heravizahed and D. Edmond. Making workflows context-aware: A way to support knowledge-intensive tasks. In *Proc. of the 5th Asian-Pacific Conference Conference on Conceptual Modelling*, pages 231 – 240, 2008.

[62] C.A.R. Hoare. *Communicating Sequential processes*. Prentice Hall, 1985.

[63] C.A.R. Hoare and He Jefing. *Unifying Theories of Programming*. Prentice Hall, 1998.

[64] C.A.R. Hoare and He Jifeng. The weakest pre-specification. *Fundamenta Informaticae*, 9:51 – 84 & 217 – 252, 1986.

[65] D. Hollingsworth. Workflow management coalition: The workflow reference model, 19-Jan-95.

[66] J. Horning, H.C. Lauer, P.M. Melliar-Smith, and B. Randell. A program structure for error detection and recovery. *Lecture Notes in Computer Science*, 16, 1974.

[67] V Huang, W.-K. & Atluri. Secureflow: A secure web-enabled workflow management system. In *Proceedings of the 4th ACM Workshop on Role-Based Access Control*, pages 83–94, 1999.

[68] P. C. K Hung. Specifying conflict of interest in web services endpoint language (wsel). 3(3):1–8, 2002.

[69] G. Hynes, V. Reynolds, and M. Hauswirth. Enabling mobility between context-aware smart spaces. In *in: International Conference on Advanced Information Networking and Applications*, pages 255–260. IEEE Computer Society, 2009.

[70] M. Jackson. *System Development*. Prentice Hall, 1983.

[71] Sushil Jajodia, Pierangela Samarati, Maria Luisa Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Trans. Database Syst.*, 26(2):214–260, 2001.

[72] P. Jalot and R.H. Campbell. Atomic action for fault-tolerance using csp. *IEEE Transactions in Software Engineering*, 12:59 – 68, 1986.

[73] Helge Janicke. Concurrent Enforcement of UCON-Style Policies. Technical report, De Montfort University, 2008.

[74] Helge Janicke, Antonio Cau, François Siewe, and Hussein Zedan. A note on the formalisation of UCON. In *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT07)*, pages 163–168, June 2007.

[75] Helge Janicke, Antonio Cau, François Siewe, and Hussein Zedan. Deriving Enforcement Mechanisms from Policies. In *Proceedings of the 8th IEEE international Workshop on Policies for Distributed Systems (POLICY2007)*, pages 161–170, June 2007.

[76] Helge Janicke, Francois Siewe, Kevin Jones, Antonio Cau, and Hussein Zedan. Analysis and Run-time Verification of Dynamic Security Policies. In Robert Ghanea-Hercock, Mark Greaves, Nick Jennings, and Simon Thompson, editors, *Proceedings of the Workshop on Defence Applications of Multi-Agent Systems (DAMAS)*, pages 55–66, Ultrecht University, The Netherlands, 2005. Springer.

[77] W. Janssen and J. Zweirs. Protocol design by layered decomposition - a compositional approachch. *Lecture Notes in Computer Science*, 571:307 – 325.

[78] O. H. Jensen and R. Milner. Bigraphs and mobile processes (revised), (2004). Tech. Rep. UCAM-CL-TR-580, University of Cambridge.

[79] J.B.D. Joshi, W.G. Aref, A. Ghafoor, and E.H. Spafford. Security models for webbased applications. *Communications of the ACM*, 44(2):38–44, 2001.

[80] M.H. Kang, J.S. Park, and J.N. Froscher. Access controlmechanisms for inter-organizational workflow. In *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, 2001.

[81] Buchmann A. Karastoyanova, D. Refflow: A model and generic approach to flexibility of web service compositions. In *Proceedings of International Conference on Information Integration and Web-based Applications and Service (iiWAS 2004)*, pages 27–29, 2004.

[82] H. Koshutanski and F. Massacci. An access control framework for business processes for web services. In *Proceedings of the 2003 ACM workshop on XML security*, pages 15–24, 2003.

[83] J. Krivine, R. Milner, and A. Troina. Stochastic bigraphs. In *Porc. of MFPS'08, 24th Conference on the Mathematical Foundations of Programming Semantics*. Elsevier, 2008.

[84] Marta Z Kwiatkowska, Gethin Norman, and Jeremy Sproston. Pctl model checking of symbolic probabilistic systems. Technical Report CSR-03-2, University of Birmingham, School of Computer Science, April 2003.

[85] Leslie Lamport. The temporal logic of actions. *ACM Trans. Program. Lang. Syst.*, 16(3):872–923, 1994.

[86] N. Leveson. *Safeware : System Safety and Computers*. Addison-Wesley Pub Co., Reading, Mass., 1995.

[87] B. Liskov and R. Scheifler. Guardians and actions: Linguistic support for robust, distributed programs. *ACM TOPLAS*, 5(3):381–404, 1983.

[88] B. Liskov and A. Snyder. Exceptional handling in clu. *IEEE Transactions in Software Engineering*, 5:546 – 558, 1979.

[89] S.W. Loke. Servive-oriented device ecology workflows. In *Proc. of the IEEE International Conference on Service-Oriented Computing*, pages 559 – 574, 2003.

[90] A. Lopes and J. L. Fiadeiro. Algebraic semantics of design abstractions for context-awareness. *Lecture Notes in Computer Science*, 3423:79–93, 2005.

[91] L. T. Ly, S. Rinderle, and P. Dadam. Integration and verification of semantic constraints in adaptive process management systems. volume 64, pages 3–23, 2008.

[92] R.D. Maddux. Fundamental study relation-algebraic semantics. *Theoretical Computer Science*, 160:1 – 85, 1996.

[93] Wil M. P. van der Aalst Maja Pesic, Helen Schonenberg. Declare: Full support for loosely-structured processes. In *Proceedings of EDOC 2007*, pages 287–300, 2007.

[94] Peter Dadam Manfred Reichert. Adeptflex - supporting dynamic changes of workflows without loosing control. 10(2):93–129, 1998.

[95] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[96] R. Milner. Pure bigraphs: structure and dynamics. *Information and Computation*, 204.

[97] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[98] R. Milner. Functions as processes. *Journal of Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[99] R. Milner. The polyadic π-calculus: A tutorial. In Friedrich L. Bauer, Wilfried Brauer, and Helmut Schwichtenberg, editors, *Logic and Algebra of Specification, Proceedings of International NATO Summer School (Marktoberdorf, Germany, 1991)*, volume 94 of *Series F*. NATO ASI, Springer, 1993. Available as Technical Report ECS-LFCS-91-180, University of Edinburgh, October 1991.

[100] R. Milner. *Communication and Mobile Systems: The π-Calculus*. Cambridge University Press, 1999.

[101] F. Moller and C.M. Tofts. A temporal calculus for communicating systems, 1989. Edinbrugh University, Technical Report, ECS-LFCS-89-104.

[102] J.H. Morris. Lambda-calculus models of programming languages, 1968. Ph.D. thesis, MIT.

[103] P.D. Mosses. *Action Semantics*. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 1992.

[104] H. R. Motahari, B. Nezhad, A. Benatallah, F. Martens, F. Curbera, and F. Casati. Semi-automated adaptation of service interactions. In *Proceedings of the 16th international conference on World Wide Web (WWW'07)*, pages 993 – 1002.

[105] M.A. Muller, U. Greiner, and E. Rahm. Agentwork: A workflow system supporting rule-based workflow adaptation. *Data & Knowledge Engineering*, 51:223–256, 2004.

[106] J. Krishnamurthy N.C. Narendra, K. Ponnalagu and R. Ramkumar. Runtime adaptation of non-functional properties of composite web services via aspect-oriented programming. In *International Conference on Service-Oriented Computing (IC-SOC)*, 2007.

[107] Paniti Netinant. Verifying semantic of system composition for an aspect-oriented approach. In *nternational Conference on System Engineering and Modeling (ICSEM 2012) IPCSIT*, volume 34, pages 50–54, 2012.

[108] M. Nivat. Nondeterministic programms: An algebraic overview. In *Information Processing'80*, pages 17–28, 1980.

[109] M. Nivat and J. Raynolds (ed.). *Algebraic methods in Semantics*. Cambridge university press, 1985.

[110] Petr Novak, Milan Rollo, Jiří Hodík, and Tomáš Vlček. Communication Security in Multi-agent Systems. In Vladimír Mařík, Jörg Müller, and Michal Pěchouček, editors, *CEEMAS*, pages 454–463. Springer, 2003.

[111] M. Nyanchama and S. Osborn. The role graph model and conflict of interest. *ACM Transactions on Information and System Security*, 2(1):3–33, 1999.

[112] OASIS. Web Services Security: SOAP Message Security 1.0 (WS-Security 2004), March 2004.

[113] OASIS. eXtensible Access Control Markup Language (XACML) Version 2.0, February 2005.

[114] OASIS. Security Assertion Markup Language (SAML) Version 2.0, February 2006.

[115] E. R. Olderog and C.A.R. Hoare. Specification orianted semantics for communication processes. *Acta Informatica*, 23:9–66, 1986.

[116] B. Pernici and A. M. Rosati. Automatic learning of repair strategies for web services. In *Proceedings of the Fifth European Conference on Web Services (ECOWS 2007)*, pages 119–128, 2007.

[117] G Pernul. Security constraint processing during multilevel secure database design. In *Proceedings of Eighth Annual IEEE Computer Security Applications Conference*, pages 229–247, 1992.

[118] R. M. Pessoa, C. Z. Calvi, J. P. Filho, C. G. de Farias, and R. Neisse. Semantic context reasoning using ontology based models. In *in: A. Pras and M. van Sinderen (Eds.), Dependable and Adaptable Networks and Services, 13th Open European Summer School and IFIP TC6.6 Workshop (EUNICE), Vol. LNCS 4606 of Lectures Notes in Computer Science, Springer Verlag, Germany*, pages 44–51, 2007.

[119] C. Petri. Concepts of net theory. In *Proc. and Symposium and Summer School on Mathematical Foundation of Computer Sciences*, pages 137 – 146, 1973.

[120] G.D. Plotkin. A structural approach to operational semantics, 1981. DAIMI-FN-19, Aarhus Univ, Denmark.

[121] M. C. B. Hennessy R. De Nicola. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[122] S.S. Ravi, J. Edward, K. Coyne, L. Hal, K. Feinstein, E. Charles, and K. Youman. Role-based access control models. *IEEE Computing*, 29(2):38–47, 1996.

[123] J.C. Raynolds. *The Craft of Programming*. Prentice Hall, 1981.

[124] G.M. Reed. *Unified Mathematical Theory for Real-Time Distributed Computing*. PhD thesis, Oxford University, 1988.

[125] Manfred Reichert and Stefanie Rinderle. On design principles for realizing adaptive service flows with bpel. In *Proc. EMISA 2006*, pages 133–146, 2006.

[126] G.-C. Roman, C. Julien, and J. Payton. Modeling adaptive behaviors in context unity. *Theoretical Computer Science*, 376 (3):185–204, 2007.

[127] M. Rosemann, J. Recker, C. Flender, and P. Ansell. Understanding context-awareness in business process design. In *Proceedings of the 17th Australian Conf. on Information Systems*, 2006.

[128] S. A. Battle S. K. Williams and J. E. Cuadrado. Protocol mediation for adap-tation in semantic web services. In *2nd European Semantic Web Conference*, pages 635–649, 2006.

[129] A. Salwicki and T. Muldner. On the algorithic properties of concurrent programs. *Lecture Notes in Computer Science*, 125, 1981.

[130] D. Sangiorgi and D. Walker. *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.

[131] D.A. Schmidt. *Denotational semantics*. Allyn and Bacon, 1986.

[132] S. Schneider. *Correctness and Communication in Real-Time Systems*. PhD thesis, Oxford University, 1990.

[133] D.A. Scott. Domains for denotational semantics. *Lecture Notes for Computer Science*, 140:577 – 613, 1982.

[134] K. Shin, Y.Chao, J. Choi, and C-W. Yoo. A workflow language for context-aware services. In *Proc. of the IEEE International Conference on Multimedia and Ubiquitous Engineering (MUE'07)*, pages 30 – 36, 2007. DOI 0-7695-2777-9/07.

[135] S. Smanchat, S. Ling, and M. Indrawan. A survey on context-aware workflow adaptations. In *Proc. of IEEE MoMM2008*, pages 414 – 417, 2008.

[136] Manfred Reichert Stefanie Rinderle and Peter Dadam. On dealing with structural conflicts between process type and instance changes. In *Proc. 2nd. Int'l Conf. Business Process Management (BPM'04)*, volume 3080 of *LNCS*, pages 274–289, 2004.

[137] J. Stoy. *Denotational Semantics*. MIT, 1977.

[138] C. Strachy. Fundamental concept in programming language. Intern Summer School in Computer Programming, 1967.

[139] R.D. Tennent. The denotational semantics of programming languages. *Communications of the ACM*, 19:437–453, 1976.

[140] R.D. Tennent. *Semantics of Programming languages*. Prentice Hall, 1991.

[141] B. Thuraisingham, C. Clifton, A. Gupta, E. Bertino, and E. Ferrari. Directions for web and e-commerce applications security. In *Proceedings of Tenth IEEE International Workshops on Enabling Technologies: Infrastructure for Col- laborative Enterprises*, pages 200–204, 2001.

[142] W.M.P. van der Aalst. Interval timed coloured petri nets and their analysis. *Lecture Notes in Computer Science*, 691:453 – 472, 1993.

[143] W.M.P. van der Aalst. The application of petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[144] V. Viera, P. Brezillon, A.C. Salgado, and P. Tedesco. A context-aware model for domain-independent context management. *Revue d'intelligence artificielle*, 16 (2):401 – 419, 2008.

[145] X. H. Wang, D. Q. Zhang, T. Gu, and H. K. Pung. Ontology based context modeling and reasoning using owl. In *Pervasive Computing and Communications Workshops, IEEE International Conference*, (2004).

[146] M. Weiser. The computer for the 21st century. *Communications*, 3(3), 1991.

[147] M. Wieland, O. Kopp, D. Nicklas, and F. Leymann. Towards context-aware workflows, 2010. Technical Report, Institute of Architecture of Application Systems, University of Stuttgart.

[148] T. Winograd. Architecture for context. *Human-Computer Interaction*, 16(2):27 – 38, 2001.

[149] H.M.W. Verbeek P.A.C. Verkoulen W.M.P. van der Aalst, T. Basten and M. Voorhoeve. Adaptive workflow: An approach based on inheritance. In *Proceedings of the IJCAI'99 Workshop on Intelligent Workflow and Process Management*, pages 36–45, 1999.

[150] H. Zedan. Safety decomposition of distributed programs. *ACM SIGPLAN Notices*, 20(8):107–112, 1985.

[151] H. Zedan. Achieving atomicity in occam. *Micoprocessing and Microprogramming*, 23:261–265, 1988.

[152] P. Zimmer. A calculus for context-awareness, 2005. Tech. rep., BRICS.